

ActiveX Data Objects 2.5 Start Page 

ADO

Version 2.5

Purpose

Microsoft ActiveX Data Objects (ADO) enable your client applications to access and manipulate data from a database server through an OLE DB provider. Its primary benefits are ease of use, high speed, low memory overhead, and a small disk footprint. ADO supports key features for building client/server and Web-based applications.

RDS

ADO also features Remote Data Service (RDS), by which you can move data from a server to a client application or Web page, manipulate the data on the client, and return updates to the server in a single round trip.

ADO MD

Microsoft ActiveX Data Objects (Multidimensional) (ADO MD) provides easy access to multidimensional data from languages such as Microsoft Visual Basic, Microsoft Visual C++, and Microsoft Visual J++. ADO MD extends Microsoft ActiveX Data Objects (ADO) to include objects specific to multidimensional data, such as the CubeDef and Cellset objects. With ADO MD you can browse multidimensional schema, query a cube, and retrieve the results.

Like ADO, ADO MD uses an underlying OLE DB provider to gain access to data. To work with ADO MD, the provider must be a multidimensional data provider (MDP) as defined by the OLE DB for OLAP specification. MDPs present data in multidimensional views as opposed to tabular data providers (TDPs) that present data in tabular views. Refer to the documentation for your OLAP OLE DB provider for more detailed information on the specific syntax and behaviors supported by your provider.

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. Because it is an object-based approach to schema manipulation, you can write code that will work against various data sources regardless of differences in their native syntaxes.

ADOX is a companion library to the core ADO objects. It exposes additional objects for creating, modifying, and deleting schema objects, such as tables and procedures. It also includes security objects to maintain users and groups and to grant and revoke permissions on objects.

Main Components of ADO 2.5

[Programmer's Guide](#)

An introduction to using ADO, RDS, ADO MD, and ADOX.

[Programmer's Reference](#)

This section of the ADO documentation contains topics for each ADO, RDS, ADO MD, and ADOX object, collection, property, dynamic property, method, event, and enumeration.

Feedback

You can send feedback about [ADO documentation](#) or [samples](#) directly to the ADO documentation team.

See Also

- [ADO Technical Articles](#)
- [OLE DB](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Guide

ADO Programmer's Guide

For an introduction to the Microsoft ActiveX Data Objects (ADO) Programmer's Guide, see the following topics:

- [Introduction](#)
- [What's New in ADO](#)
- [Prerequisites](#)
- [The ADO Family of Libraries](#)
- [The Role of ADO in Microsoft Data Access](#)
- [ADO Task Table](#)
- [ADO Technology Table](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Guide

Introduction

The ADO Programmer's Guide has been created to assist developers who are new to ADO by giving them a thorough introduction to the technology. This guide describes the libraries of the ADO family and uses sample code in a variety of languages to explain how to use the libraries, best practices for using the libraries, and tips and tricks for maximizing the performance of your ADO application.

The ADO Programmer's Guide contains the following sections and chapters:

- [Section I: ActiveX Data Objects \(ADO\)](#)
 - [Chapter 1: ADO Fundamentals](#)
 - [Chapter 2: Getting Data](#)
 - [Chapter 3: Examining Data](#)
 - [Chapter 4: Editing Data](#)
 - [Chapter 5: Updating and Persisting Data](#)
 - [Chapter 6: Error Handling](#)
 - [Chapter 7: Handling ADO Events](#)
 - [Chapter 8: Understanding Cursors and Locks](#)
 - [Chapter 9: Data Shaping](#)
 - [Chapter 10: Records and Streams](#)
- [Section II: Remote Data Service \(RDS\)](#)
 - [Chapter 11: RDS Fundamentals](#)
 - [Chapter 12: RDS Tutorial](#)
 - [Chapter 13: RDS Usage and Security](#)
- [Section III: ActiveX Data Objects \(Multidimensional\) \(ADO MD\)](#)
 - [Chapter 14: ADO MD Fundamentals](#)
- [Section IV: ActiveX Data Objects Extensions for Data Definition Language and Security \(ADOX\)](#)
 - [Chapter 15: ADOX Fundamentals](#)
- [Section V: Appendixes](#)
 - [Appendix A: Providers](#)
 - [Appendix B: ADO Errors](#)
 - [Appendix C: Programming with ADO](#)
 - [Appendix D: ADO Samples](#)

- [ADO Glossary](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Guide

What's New in ADO

The following new features and enhanced documentation are included in the ADO 2.5 release. This list covers ADO, ADO MD, and ADOX.

New Features

[Records and Streams](#)

This release of ADO introduces the [Record](#) object, which can represent and manage things like directories and files in a file system, and folders and messages in an e-mail system. A **Record** can also represent a row in a [Recordset](#), although **Record** and **Recordset** objects have different methods and properties.

The new [Stream](#) object provides the means to read, write, and manage the binary stream of bytes or text that comprise a file or message stream.

[URL Usage](#)

This release also introduces the use of Uniform Resource Locators (URLs), as an alternative to connection strings and command text, to name data store objects. URLs may be used with the existing [Connection](#) and **Recordset** objects, as well as with the new **Record** and **Stream** objects.

With this release, ADO supports OLE DB providers that recognize their own URL schemes. For example, the [OLE DB Provider for Internet Publishing](#), which accesses the Windows 2000 file system, recognizes the existing HTTP scheme.

[Special Fields for Document Source Providers](#)

A special class of providers, called *document source* providers, manage folders and documents. When a **Record** object represents a document, or a **Recordset** object represents a folder of documents, the document source provider populates those objects with a unique set of fields that describe characteristics of the document. These fields constitute a *resource* **Record** or **Recordset**.

New Reference Topics

The following new properties are included in this release.

Property	Description
Charset	Indicates the character set into which the contents of a text Stream object should be translated.
EOS	Indicates whether the current position is at the end of the stream.
LineSeparator	Indicates the binary character to be used as the line separator in text Stream objects.
Mode	Indicates the available permissions for modifying data in a Connection , Record , or Stream object.
ParentURL	Indicates an absolute URL string that points to the parent Record of the current Record object.
Position	Indicates the current position within a Stream object.
RecordType	Indicates the type of Record object.
Size	Indicates the size of the stream in number of bytes.
Source	Indicates the entity represented by the Record object.
State	Indicates for all applicable objects whether the state of the object is open or closed. Indicates for all applicable objects executing an asynchronous method, whether the current state of the object is connecting, executing, or retrieving.
Type	Indicates the type of data contained in the Stream object (binary or text).

The following new methods are included in this release.

Method	Description
CopyRecord	Copies a file or directory, and its contents, to another location.
	Copies the specified number of characters or bytes

CopyTo	(depending on Type) in the Stream object to another Stream object.
DeleteRecord	Deletes a file or directory, and all its subdirectories.
Flush	Forces the contents of the Stream object remaining in the ADO buffer to the underlying object with which the Stream object is associated.
GetChildren	Returns a Recordset whose rows represent the files and subdirectories in the directory represented by this Record .
LoadFromFile	Loads the contents of an existing file into a Stream object.
MoveRecord	Moves a file, or a directory and its contents, to another location.
Open	Opens an existing Record object, or creates a new file or directory.
Open	Opens a Stream object to manipulate streams of binary or text data.
Read	Reads a specified number of bytes from a binary Stream object.
ReadText	Reads specified number of characters from a text Stream object.
SaveToFile	Saves the binary contents of a Stream to a file.
SetEOS	Sets the position that is the end of the stream.
SkipLine	Skips one entire line when reading a text Stream object.
Write	Writes binary data to a Stream object.
WriteText	Writes a specified text string to a Stream object.

New and Enhanced Documentation

[Code Example Topics](#)

The examples have been expanded to contain code examples written in Microsoft Visual C++® and Microsoft Visual J++®. You can copy and paste these code examples into your editor.

Provider Topics

A new topic is included that explains how to use ADO with the [OLE DB Provider for Internet Publishing](#).

Programming with ADO

This new section contains tips and tricks for using ADO with various programming languages. It contains the existing syntax indexes for the Visual C++ Extensions for ADO and ADO/WFC, as well as new information specific to developers using Microsoft Visual Basic®, Microsoft Visual Basic® Scripting Edition, Microsoft JScript®, Microsoft Visual C++, or Microsoft Visual J++.

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Guide

Prerequisites

The *ADO Programmer's Guide* will prove useful to developers with a wide variety of backgrounds. At a minimum, readers should have an intermediate level of experience in developing applications with Microsoft Visual Basic, because most of the examples in the guide are written in this language. Other examples are written in Microsoft Visual C++; Java; Visual Basic, Scripting Edition (VBScript); and Microsoft JScript.

Because ADO is used for accessing data from a variety of sources, readers might also need some understanding of fundamental relational database management system concepts, online analytical processing (OLAP) concepts, and basic familiarity with the Internet and Internet protocols.

ADO is a part of the Microsoft Data Access (UDA) strategy. (For more information about UDA, see "[The Role of ADO in Microsoft Data Access](#)," later in this chapter.) As such, it interoperates with the OLE DB technology. OLE DB is based on the Microsoft Component Object Model (COM). Therefore, familiarity with COM can also be useful for understanding some of the more advanced concepts in ADO.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Guide

The ADO Family of Libraries

Three major libraries make up the ADO family: ADO (including RDS), ADO MD, and ADOX.

ADO

ADO enables your client applications to access and manipulate data from a database server through an OLE DB provider. The primary benefits of ADO are ease of use, high speed, low memory overhead, and a small disk footprint. ADO supports key features for building client/server and Web-based applications.

ADO also features Remote Data Service (RDS), by which you can move data from a server to a client application or Web page, manipulate the data on the client, and return updates to the server in a single round trip.

ADO MD

Microsoft ActiveX Data Objects (Multidimensional) (ADO MD) provides easy access to multidimensional data from languages such as Microsoft Visual Basic, Microsoft Visual C++, and Microsoft Visual J++. ADO MD extends ADO to include objects specific to multidimensional data, such as the **CubeDef** and **Cellset** objects. With ADO MD you can browse multidimensional schema, query a cube, and retrieve the results.

Like ADO, ADO MD uses an underlying OLE DB provider to gain access to data. To work with ADO MD, the provider must be a multidimensional data provider (MDP) as defined by the OLE DB for OLAP specification. MDPs present data in multidimensional views as opposed to tabular data providers (TDPs) that present data in tabular views. Refer to the documentation for your OLE DB for OLAP provider for more detailed information about the specific syntax and behaviors supported by your provider.

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. Because it is an object-based approach to schema manipulation, you can write code that will work against various data sources regardless of differences in their native syntaxes.

ADOX is a companion library to the core ADO objects. It exposes additional objects for creating, modifying, and deleting schema objects, such as tables and procedures. It also includes security objects to maintain users and groups, and to grant and revoke permissions on objects.

See Also

[Section I: ActiveX Data Objects \(ADO\)](#) | [Section II: Remote Data Service \(RDS\)](#) | [Section III: ADO \(Multidimensional\) \(ADO MD\)](#) | [Section IV: ADO Extensions for Data Definition Language and Security \(ADOX\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Guide

The Role of ADO in Microsoft Data Access

The Microsoft Data Access Components (MDAC) provide data access that is independent of data stores, tools, and languages. It provides a high-level, easy-to-use interface, and a low-level, high-performance interface to practically any data store available. You can use this flexibility to integrate diverse data stores and use your choice of tools, applications, and platform services to create the right solutions for your needs. These technologies provide the basic framework for general-purpose data access in Microsoft Windows operating systems.

There are three primary technologies in MDAC. ActiveX Data Objects (ADO) is a high-level, easy-to-use interface to OLE DB. OLE DB is a low-level, high-performance interface to a variety of data stores. ADO and OLE DB both can work with relational (tabular) and nonrelational (hierarchical or stream) data. Finally, Open Database Connectivity (ODBC) is another low-level, high-performance interface that is designed specifically for relational data stores.

ADO provides a layer of abstraction between your client or middle-tier application and the low-level OLE DB interfaces. ADO uses a small set of Automation objects to provide a simple and efficient interface to OLE DB. This interface makes ADO the perfect choice for developers in higher level languages, such as Visual Basic and even VBScript, who want to access data without having to learn the intricacies of COM and OLE DB.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Guide

ADO Task Table

The following table lists programming tasks contained in the *ADO Programmer's Guide* and provides references for each task. These references can be textual descriptions or code examples in which you can find information about the ADO feature that performs the task.

ADO Task	References
Connecting to a data provider	Making a Connection
Executing commands or calling stored procedures	Using the Command Object
Opening a Recordset	The Recordset Object Open Method
Determining the size of a Recordset	Counting Rows and The Limits of a Recordset
Moving to a specific record	Navigating Through the Data
Accessing column values	The Fields Collection
Searching for data	Working with Recordsets
Modifying data and changing values	Editing Existing Records
Adding new data	Adding Records
Deleting or removing data	Deleting Records Using the Delete Method
Posting changes to the data source	Updating Data
Beginning, committing, and rolling back transactions	Transaction Processing
Saving records to a file (XML or binary)	Persisting Data
Handling errors	ADO Errors
Handling events, asynchronous programming	ADO Event Handler Summary

Choosing cursor location and type	Types of Cursors
Choosing lock types	Types of Locks
Returning related records in a Recordset	Data Shaping Summary
Accessing semi-structured data	Chapter 10: Records and Streams
Publishing to IIS	Using ADO for Internet Publishing

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Guide

ADO Technology Table

The following table lists the Microsoft products, tools, and technologies discussed in the *ADO Programmer's Guide*. It provides links, wherever possible, to relevant topics in the guide.

Microsoft Product or Technology	References
	Working with Recordsets
	Chapter 15: ADOX Fundamentals
Access/Jet	Microsoft Jet and Replication Objects OLE DB Provider for Microsoft Jet
Active Directory Service Interfaces	Microsoft OLE DB Provider for Microsoft Active Directory Service Prerequisites Marking Business Objects as Safe for Scripting Registering Business Objects on the Client for Use with DCOM
COM/DCOM	Setting DCOM Stream Marshaling Format Enabling a DLL to Run on DCOM Running Business Objects in Component Services
FrontPage	Microsoft OLE DB Provider for Internet Publishing
Indexing Service	Microsoft OLE DB Provider for Microsoft Indexing Service

Internet Explorer

[Internet Explorer Error Codes](#)
[Chapter 10: Records and Streams](#)

[Streams and Persistence](#)

[Using ADO for Internet Publishing](#)

[Solutions for Remote Data Access](#)

[Configuring Virtual Servers on IIS](#)

Internet Information Services

[Specifying Threads Per Processor on IIS](#)

[Securing RDS Applications](#)

["Internet Server Error: Access Denied"](#)

[Microsoft OLE DB Provider for Internet Publishing](#)

[Internet Information Services Error Codes](#)

[Handling Errors in Other Languages](#)

JScript

[JScript ADO Programming](#)

[ADO Code Examples in Microsoft JScript](#)

[The Role of ADO in Microsoft Data Access](#)

[Using the Connection Object](#)

ODBC

[Using RDS with ODBC Connection Pooling](#)

[Microsoft OLE DB Provider for ODBC](#)

	<u>The Role of ADO in Microsoft Data Access</u>
OLE DB	<u>OLE DB Providers</u> <u>Appendix A: Providers</u> <u>Provider Errors</u>
Oracle	<u>Microsoft OLE DB Provider for Oracle</u> <u>Controlling Transactions</u> <u>Calling a Stored Procedure with a Command</u> <u>Counting Rows</u> <u>Forward-Only Cursors</u>
SQL Server 2000	<u>Command Streams</u> <u>Ensuring Sufficient TempDB Space</u> <u>Minimizing Log File Space Usage</u> <u>Microsoft OLE DB Provider for SQL Server</u>
Transaction Server	<u>Running Business Objects in Component Services</u> <u>Handling Errors in Other Languages</u> <u>Visual Basic for Applications Functions</u> <u>Command Streams</u> <u>Solutions for Remote Data Access</u>
VBScript	<u>RDS Scenario</u>

[RDS Tutorial \(VBScript\)](#)

[VBScript ADO Programming](#)

[ADO Code Examples in Microsoft Visual Basic Scripting Edition](#)

[Errors](#)

[ADO Errors](#)

[ADO Event Instantiation By Language](#)

[Visual Basic for Applications Functions](#)

Visual Basic

[Chapter 12: RDS Tutorial](#)

[Using ADO with Microsoft Visual Basic](#)

[ADO Code Examples in Microsoft Visual Basic](#)

[Handling Errors in Other Languages](#)

[ADO Event Instantiation By Language](#)

Visual C++

[Using ADO with Microsoft Visual C++](#)

[ADO Code Examples in Microsoft Visual C++](#)

[Handling Errors in Other Languages](#)

[ADO Event Instantiation By Language](#)

Visual J++

[RDS Tutorial \(Visual J++\)](#)

[Using ADO with Microsoft Visual J++](#)

[ADO Code Examples in Microsoft Visual J++](#)

Visual Studio

[Appendix D: ADO Samples](#)
[System Requirements for the Address Book Application](#)

[Granting Guest Privileges to a Web Server Computer](#)

Windows 2000

[Registering a Custom Business Object](#)

[Securing RDS Applications](#)

[Configuring RDS on Windows 2000](#)

[Persisting Records in XML Format](#)

[Chapter 10: Records and Streams](#)

XML

[Command Streams](#)

[Retrieving Resultsets into Streams](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Section I: ActiveX Data Objects (ADO)

This section contains the following chapters:

- [Chapter 1: ADO Fundamentals](#)
- [Chapter 2: Getting Data](#)
- [Chapter 3: Examining Data](#)
- [Chapter 4: Editing Data](#)
- [Chapter 5: Updating and Persisting Data](#)
- [Chapter 6: Error Handling](#)
- [Chapter 7: Handling ADO Events](#)
- [Chapter 8: Understanding Cursors and Locks](#)
- [Chapter 9: Data Shaping](#)
- [Chapter 10: Records and Streams](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Chapter 1: ADO Fundamentals

This chapter is an introduction to the ADO library. It discusses what you can do with ADO, reviews the objects in the ADO hierarchy, and presents a simple ADO application that uses many of the ADO objects to retrieve, edit, and update data from a data source. Finally, this chapter covers two issues that are important to understand for writing ADO applications: OLE DB providers and errors.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

What You Can Do With ADO

ADO is designed to provide developers with a powerful, logical object model for programmatically accessing, editing, and updating a wide variety of data sources through OLE DB system interfaces. The most common usage of ADO is to query a table or tables in a relational database, retrieve and display the results in an application, and perhaps allow users to make and save changes to the data. Other things that can be done programmatically with ADO include:

- Querying a database using SQL and displaying the results.
- Accessing information in a file store over the Internet.
- Manipulating messages and folders in an e-mail system.
- Saving data from a database into an XML file.
- Allowing a user to review and make changes to data in database tables.
- Creating and reusing parameterized database commands.
- Executing stored procedures.
- Dynamically creating a flexible structure, called a **Recordset**, to hold, navigate, and manipulate data.
- Performing transactional database operations.
- Filtering and sorting local copies of database information based on run-time criteria.
- Creating and manipulating hierarchical results from databases.
- Binding database fields to data-aware components.
- Creating remote, disconnected **Recordsets**.

ADO must expose a wide variety of options and settings in order to provide such flexibility. Therefore it's important to take a methodical approach to learning how to use ADO in an application, breaking down each of your goals into manageable pieces.

Four primary operations are involved in most ADO programs: getting data, examining data, editing data, and updating data. The next four chapters examine each of these operations in more detail.

Before proceeding, familiarize yourself with the objects in the [ADO Object Model](#). Then review [HelloData: A Simple ADO Application](#). This application is written in Visual Basic and performs each of the four primary ADO operations.

© 1998-2002 Microsoft Corporation. All rights reserved.

ADO 2.5 

The ADO Object Model

ADO requires only nine objects and four collections to provide its entire functionality. The following table introduces them.

Object or Collection	Description
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. Depending on the functionality supported by the provider, some collections, methods, or properties of a Connection object may not be available.
Command object	Used to define a specific command, such as a SQL query, intended to run against a data source.
Recordset object	Represents the entire set of records from a base table or the results of an executed command. All Recordset objects consist of records (rows) and fields (columns).
Record object	Represents a single row of data, either from a Recordset or from the provider. This record could represent a database record or some other type of object such as a file or directory, depending upon your provider.
Stream object	Represents a stream of binary or text data. For example, an XML document can be loaded into a stream for command input or returned from certain providers as the results of a query. A Stream object can be used to

	manipulate fields or records containing these streams of data.
Parameter object	Represents a parameter or argument associated with a Command object, based on a parameterized query or stored procedure.
Field object	Represents a column of data with a common data type. Each Field object corresponds to a column in the Recordset .
Property object	Represents a characteristic of an ADO object that is defined by the provider. ADO objects have two types of properties: built-in and dynamic. Built-in properties are those properties implemented in ADO and immediately available to any new object. The Property object is a container for dynamic properties, defined by the underlying provider.
Error object	Contains details about data access errors that pertain to a single operation involving the provider.
Fields collection	Contains all the Field objects of a Recordset or Record object.
Properties collection	Contains all the Property objects for a specific instance of an object.
Parameters collection	Contains all the Parameter objects of a Command object.
Errors collection	Contains all the Error objects created in response to a single provider-related failure.

The following figures show the ADO objects and their collections. Click an object or collection for more information from the ADO Programmer's Reference.

Connection

<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>

<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>

<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

ADO 2.5 

HelloData: A Simple ADO Application

To lay the groundwork for an exploration of the ADO library, consider a simple ADO application called "HelloData." HelloData steps through each of the four major ADO operations (getting, examining, editing, and updating data). In order to focus on the fundamentals of ADO and prevent code clutter, minimal error handling is done in the example.

The application queries the Northwind sample database that is included with Microsoft SQL Server 2000.

To run HelloData

1. Create a new Standard Executable Visual Basic Project that references the ADO 2.5 library.
2. Create four command buttons at the top of the form, setting the **Name** and **Caption** properties to the values shown in the table below.
3. Below the buttons, add a **Microsoft DataGrid Control** (Msdatgrd.ocx). The Msdatgrd.ocx file comes with Visual Basic and is located in your \windows\system32 or \winnt\system32 directory. To add the DataGrid control to your Visual Basic toolbox pane, select **Components...** from the **Project** menu. Then check the box next to "Microsoft DataGrid Control 6.0 (SP3) (OLEDB)" and click **OK**. To add the control to the project, drag the DataGrid control from the Toolbox to the Visual Basic form.
4. Create a **TextBox** on the form below the grid and set its properties as shown in the table. The form should look similar to the following figure when you are finished.
5. Finally, copy the code listed in "[HelloData Code](#)" and paste it into the code editor window of the form. Press **F5** to run the code.

Note In the following example, and throughout the guide, the user id "MyId" with a password of "123aBc" is used to authenticate against the server. You should substitute these values with valid logon credentials for your server. Also, substitute the "MyServer" value with the name of your server.

For a detailed description of the code, see "[HelloData Details](#)."



Control Type	Property	Value
Form	Name	Form1
	Height	6500
	Width	6500
MS DataGrid	Name	grdDisplay1
TextBox	Name	txtDisplay1
	Multiline	true
Command Button	Name	cmdGetData
	Caption	Get Data
Command Button	Name	cmdExamineData
	Caption	Examine Data
Command Button	Name	cmdEditData
	Caption	Edit Data
Command Button	Name	cmdUpdateData
	Caption	Update Data

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

HelloData Details

The HelloData application steps through the basic operations of a typical ADO application: getting, examining, editing, and updating data. When you start the application, click the first button, **Get Data**. This will run the GetData() subroutine.

GetData

GetData places a valid connection string into a module-level variable, *m_sConnStr*. For more information about connection strings, see [Creating the Connection String](#).

Assign an error handler using a Visual Basic **OnError** statement. For more information about error handling in ADO, see [Chapter 6: Error Handling](#). A new **Connection** object is created, and the **CursorLocation** property is set to **adUseClient** because the HelloData example creates a *disconnected Recordset*. This means that once the data has been fetched from the data source, the physical connection with the data source is broken, but you can still work with the data that is cached locally in your **Recordset** object.

After the connection has been opened, assign a SQL string to a variable (*sSQL*). Then instantiate a new **Recordset** object, *m_oRecordset1*. In the next line of code, open the **Recordset** over the existing **Connection**, passing in *sSQL* as the source of the **Recordset**. You assist ADO in making the determination that the SQL string you have passed as the source for the **Recordset** is a textual definition of a command by passing **adCmdText** in the final argument to the **Recordset Open** method. This line also sets the **LockType** and **CursorType** associated with the **Recordset**.

The next line of code sets the **MarshalOptions** property equal to **adMarshalModifiedOnly**. **MarshalOptions** indicates which records should be [marshaled](#) to the [middle tier](#) (or [web server](#)). For more information about marshaling, see the COM documentation. When using **adMarshalModifiedOnly** with a client-side cursor (**CursorLocation** = **adUseClient**), only records that have been modified on the client are written back to the middle tier. Setting **MarshalOptions** to **adMarshalModifiedOnly** can improve performance because fewer rows are marshaled.

Next, disconnect the **Recordset** by setting its **ActiveConnection** property equal to **Nothing**. For more information, see [Disconnecting and Reconnecting the Recordset](#) in Chapter 5: Updating and Persisting Data.

Close the connection to the data source and destroy the existing **Connection** object, thereby releasing the resources it consumed.

The final step is to set the **Recordset** as the **DataSource** for the Microsoft DataBound Grid Control on the form so that you can easily display the data from the **Recordset** on the form.

Click the second button, **Examine Data**. This runs the ExamineData subroutine.

ExamineData

ExamineData uses various methods and properties of the **Recordset** object to display information about the data in the **Recordset**. It reports the number of records by using the **RecordCount** property. It loops through the **Recordset** and prints the value of the **AbsolutePosition** property in the display text box on the form. Also while in the loop, the value of the **Bookmark** property for the third record is placed into a variant variable, *vBookmark*, for later use.

The routine navigates directly back to the third record using the bookmark variable that it stored earlier. The routine calls the WalkFields subroutine, which loops through the **Fields** collection of the **Recordset** and displays details about each **Field** in the collection.

Finally, ExamineData uses the **Filter** property of the **Recordset** to screen for only those records with a CategoryId equal to 2. The result of applying this filter is immediately visible in the display grid on the form.

For more information about the functionality shown in the ExamineData subroutine, see [Chapter 3: Examining Data](#).

Next, click the third button, **Edit Data**. This will run the EditData subroutine.

EditData

When the code enters the EditData subroutine, the **Recordset** is still filtered on CategoryId equal to 2, so only those items that meet the filter criteria are visible. It first loops through the **Recordset** and increases the price of each visible item in the **Recordset** by 10 percent. The value of the **Price** field is changed by setting the **Value** property for that field equal to a new, valid amount.

Remember that the **Recordset** is disconnected from the data source. The changes made in EditData are made only to the locally cached copy of the data. For more information, see [Chapter 4: Editing Data](#).

The changes will not be made on the data source until you click the fourth button, **Update Data**. This will run the UpdateData subroutine.

UpdateData

UpdateData first removes the filter that has been applied to the **Recordset**. The code removes and resets `m_oRecordset1` as the **DataSource** for the Microsoft Bound DataGrid on the form so that the unfiltered **Recordset** appears in the grid.

The code then checks to see whether you can move backward in the **Recordset** by using the **Supports** method with the **adMovePrevious** argument.

The routine moves to the first record using the **MoveFirst** method and displays the field's original and current values, using the **OriginalValue** and **Value** properties of the **Field** object. These properties, along with the **UnderlyingValue** property (not used here), are discussed in [Chapter 5: Updating and Persisting Data](#).

Next, a new **Connection** object is created and used to reestablish a connection to the data source. You reconnect the **Recordset** to the data source by setting the new **Connection** as the **ActiveConnection** for the **Recordset**. To send the updates to the server, the code calls **UpdateBatch** on the **Recordset**.

If the batch update succeeds, a module-level flag variable, `m_flgPriceUpdated`, is set to True. This will remind you later to clean up all changes made to the database.

Finally, the code moves back to the first record in the **Recordset** and displays the original and current values. The values are the same after the call to **UpdateBatch**.

For more detailed information about updating data, including what to do when data on the server changes while your **Recordset** is disconnected, see [Chapter 5: Updating and Persisting Data](#).

Form_Unload

The Form_Unload subroutine is important for several reasons. First, because this is a sample application, Form_Unload cleans up the changes made to the database before the application exits. Second, the code shows how a command can be executed directly from an open **Connection** object using the **Execute** method. Finally, it shows an example of executing a non-row-returning query (an UPDATE query) against the data source.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

HelloData Code

```
'BeginHelloData
Option Explicit

Dim m_oRecordset As ADODB.Recordset
Dim m_sConnStr As String
Dim m_flgPriceUpdated As Boolean

Private Sub cmdGetData_Click()
    GetData

    If Not m_oRecordset Is Nothing Then
        If m_oRecordset.State = adStateOpen Then
            ' Set the proper states for the buttons.
            cmdGetData.Enabled = False
            cmdExamineData.Enabled = True
        End If
    End If
End Sub

Private Sub cmdExamineData_Click()
    ExamineData
End Sub

Private Sub cmdEditData_Click()
    EditData
End Sub

Private Sub cmdUpdateData_Click()
    UpdateData

    ' Set the proper states for the buttons.
    cmdUpdateData.Enabled = False
End Sub

Private Sub GetData()
    On Error GoTo GetDataError

    Dim sSQL As String
    Dim oConnection1 As ADODB.Connection

    m_sConnStr = "Provider='SQLOLEDB';Data Source='MySqlServer';" &
        "Initial Catalog='Northwind';Integrated Security='SS

    ' Create and Open the Connection object.
```

```

Set oConnection1 = New ADODB.Connection
oConnection1.CursorLocation = adUseClient
oConnection1.Open m_sConnStr

sSQL = "SELECT ProductID, ProductName, CategoryID, UnitPrice " &
      "FROM Products"

' Create and Open the Recordset object.
Set m_oRecordset = New ADODB.Recordset
m_oRecordset.Open sSQL, oConnection1, adOpenStatic, _
                  adLockBatchOptimistic, adCmdText

m_oRecordset.MarshalOptions = adMarshalModifiedOnly

' Disconnect the Recordset.
Set m_oRecordset.ActiveConnection = Nothing
oConnection1.Close
Set oConnection1 = Nothing

' Bind Recordset to the DataGrid for display.
Set grdDisplay1.DataSource = m_oRecordset

Exit Sub

GetDataError:
If Err <> 0 Then
    If oConnection1 Is Nothing Then
        HandleErrs "GetData", m_oRecordset.ActiveConnection
    Else
        HandleErrs "GetData", oConnection1
    End If
End If

If Not oConnection1 Is Nothing Then
    If oConnection1.State = adStateOpen Then oConnection1.Close
    Set oConnection1 = Nothing
End If
End Sub

Private Sub ExamineData()
    On Err GoTo ExamineDataErr

    Dim iNumRecords As Integer
    Dim vBookmark As Variant

    iNumRecords = m_oRecordset.RecordCount

    DisplayMsg "There are " & CStr(iNumRecords) & _
              " records in the current Recordset."

```

```

' Loop through the Recordset and print the
' value of the AbsolutePosition property.
DisplayMsg "***** Start AbsolutePosition Loop *****"

Do While Not m_oRecordset.EOF
    ' Store the bookmark for the 3rd record,
    ' for demo purposes.
    If m_oRecordset.AbsolutePosition = 3 Then _
        vBookmark = m_oRecordset.Bookmark

        DisplayMsg m_oRecordset.AbsolutePosition

        m_oRecordset.MoveNext
Loop

DisplayMsg "***** End AbsolutePosition Loop *****" & vbCrLf

' Use our bookmark to move back to 3rd record.
m_oRecordset.Bookmark = vBookmark
MsgBox vbCrLf & "Moved back to position " & _
        m_oRecordset.AbsolutePosition & " using bookmark.", , _
        "Hello Data"

' Display meta-data about each field. See WalkFields() sub.
Call WalkFields

' Apply a filter on the type field.
MsgBox "Filtering on type field. (CategoryID=2)", _
        vbOKOnly, "Hello Data"

m_oRecordset.Filter = "CategoryID=2"

' Set the proper states for the buttons.
cmdExamineData.Enabled = False
cmdEditData.Enabled = True

Exit Sub

ExamineDataErr:
    HandleErrs "ExamineData", m_oRecordset.ActiveConnection
End Sub

Private Sub EditData()
    On Error GoTo EditDataErr

    'Recordset still filtered on CategoryID=2.
    'Increase price by 10% for filtered records.
    MsgBox "Increasing unit price by 10%" & vbCrLf & _
        "for all records with CategoryID = 2.", , "Hello Data"

```

```

m_oRecordset.MoveFirst

Dim cVal As Currency
Do While Not m_oRecordset.EOF
    cVal = m_oRecordset.Fields("UnitPrice").Value
    m_oRecordset.Fields("UnitPrice").Value = (cVal * 1.1)
    m_oRecordset.MoveNext
Loop

' Set the proper states for the buttons.
cmdEditData.Enabled = False
cmdUpdateData.Enabled = True

Exit Sub

EditDataErr:
    HandleErrs "EditData", m_oRecordset.ActiveConnection
End Sub

Private Sub UpdateData()
    On Error GoTo UpdateDataErr

    Dim oConnection2 As New ADODB.Connection

    MsgBox "Removing Filter (adFilterNone).", , "Hello Data"
    m_oRecordset.Filter = adFilterNone

    Set grdDisplay1.DataSource = Nothing
    Set grdDisplay1.DataSource = m_oRecordset

    MsgBox "Applying Filter (adFilterPendingRecords).", , "Hello Dat
    m_oRecordset.Filter = adFilterPendingRecords

    Set grdDisplay1.DataSource = Nothing
    Set grdDisplay1.DataSource = m_oRecordset

    DisplayMsg "*** PRE-UpdateBatch values for 'UnitPrice' field. **

    ' Display Value, UnderlyingValue, and OriginalValue for
    ' type field in first record.
    If m_oRecordset.Supports(adMovePrevious) Then
        m_oRecordset.MoveFirst
        DisplayMsg "OriginalValue = " & _
            m_oRecordset.Fields("UnitPrice").OriginalValue
        DisplayMsg "Value = " & _
            m_oRecordset.Fields("UnitPrice").Value
    End If

    oConnection2.ConnectionString = m_sConnStr

```

```

oConnection2.Open

Set m_oRecordset.ActiveConnection = oConnection2
m_oRecordset.UpdateBatch

m_flgPriceUpdated = True

DisplayMsg "*** POST-UpdateBatch values for 'UnitPrice' field **

If m_oRecordset.Supports(adMovePrevious) Then
    m_oRecordset.MoveFirst
    DisplayMsg "OriginalValue  = " & _
        m_oRecordset.Fields("UnitPrice").OriginalValue
    DisplayMsg "Value          = " & _
        m_oRecordset.Fields("UnitPrice").Value
End If

MsgBox "See value comparisons in txtDisplay.", , _
    "Hello Data"

'Clean up
oConnection2.Close
Set oConnection2 = Nothing
Exit Sub

UpdateDataErr:
If Err <> 0 Then
    HandleErrs "UpdateData", oConnection2
End If

If Not oConnection2 Is Nothing Then
    If oConnection2.State = adStateOpen Then oConnection2.Close
    Set oConnection2 = Nothing
End If
End Sub

Private Sub WalkFields()
    On Error GoTo WalkFieldsErr

    Dim iFldCnt As Integer
    Dim oFields As ADODB.Fields
    Dim oField As ADODB.Field
    Dim sMsg As String

    Set oFields = m_oRecordset.Fields

    DisplayMsg "***** BEGIN FIELDS WALK *****"

    For iFldCnt = 0 To (oFields.Count - 1)

```

```

    Set oField = oFields(iFldCnt)
    sMsg = ""
    sMsg = sMsg & oField.Name
    sMsg = sMsg & vbTab & "Type: " & GetTypeAsString(oField.Type)
    sMsg = sMsg & vbTab & "Defined Size: " & oField.DefinedSize
    sMsg = sMsg & vbTab & "Actual Size: " & oField.ActualSize

    grdDisplay1.SelStartCol = iFldCnt
    grdDisplay1.SelEndCol = iFldCnt
    DisplayMsg sMsg
    MsgBox sMsg, , "Hello Data"
Next iFldCnt

DisplayMsg "***** END FIELDS WALK *****" & vbCrLf

'Clean up
Set oField = Nothing
Set oFields = Nothing
Exit Sub

WalkFieldsErr:
Set oField = Nothing
Set oFields = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub

Private Function GetTypeAsString(dtType As ADODB.DataTypeEnum) As St
' To save space, we are only checking for data types
' that we know are present.
Select Case dtType
    Case adChar
        GetTypeAsString = "adChar"
    Case adVarChar
        GetTypeAsString = "adVarChar"
    Case adVarWChar
        GetTypeAsString = "adVarWChar"
    Case adCurrency
        GetTypeAsString = "adCurrency"
    Case adInteger
        GetTypeAsString = "adInteger"
End Select
End Function

Private Sub HandleErrs(sSource As String, ByRef m_oConnection As ADC
DisplayMsg "ADO (OLE) ERROR IN " & sSource
DisplayMsg vbTab & "Error: " & Err.Number
DisplayMsg vbTab & "Description: " & Err.Description

```

```

DisplayMsg vbTab & "Source: " & Err.Source

If Not m_oConnection Is Nothing Then
    If m_oConnection.Errors.Count <> 0 Then
        DisplayMsg "PROVIDER ERROR"
        Dim oError1 As ADODB.Error
        For Each oError1 In m_oConnection.Errors
            DisplayMsg vbTab & "Error: " & oError1.Number
            DisplayMsg vbTab & "Description: " & oError1.Descrip
            DisplayMsg vbTab & "Source: " & oError1.Source
            DisplayMsg vbTab & "Native Error:" & oError1.NativeE
            DisplayMsg vbTab & "SQL State: " & oError1.SQLState
        Next oError1
        m_oConnection.Errors.Clear
        Set oError1 = Nothing
    End If
End If

MsgBox "Error(s) occurred. See txtDisplay1 for specific informat
"Hello Data"

Err.Clear
End Sub

Private Sub DisplayMsg(sText As String)
    txtDisplay1.Text = (txtDisplay1.Text & vbCrLf & sText)
End Sub

Private Sub Form_Resize()
    grdDisplay1.Move 100, 700, Me.ScaleWidth - 200, (Me.ScaleHeight
    txtDisplay1.Move 100, grdDisplay1.Top + grdDisplay1.Height + 100
    Me.ScaleWidth - 200, (Me.ScaleHeight - 1000) / 2
End Sub

Private Sub Form_Load()
    cmdGetData.Enabled = True
    cmdExamineData.Enabled = False
    cmdEditData.Enabled = False
    cmdUpdateData.Enabled = False

    grdDisplay1.AllowAddNew = False
    grdDisplay1.AllowDelete = False
    grdDisplay1.AllowUpdate = False
    m_flgPriceUpdated = False
End Sub

Private Sub Form_Unload(Cancel As Integer)
    On Error GoTo ErrHandler:

```

```

Dim oConnection3 As New ADODB.Connection
Dim sSQL As String
Dim lAffected As Long

' Undo the changes we've made to the database on the server.
If m_flgPriceUpdated Then
    sSQL = "UPDATE Products SET UnitPrice=(UnitPrice/1.1) " & _
        "WHERE CategoryID=2"
    oConnection3.Open m_sConnStr
    oConnection3.Execute sSQL, lAffected, adCmdText

    MsgBox "Restored prices for " & CStr(lAffected) & _
        " records affected.", , "Hello Data"
End If

'Clean up
oConnection3.Close
Set oConnection3 = Nothing
m_oRecordset.Close
Set m_oRecordset = Nothing
Exit Sub

ErrorHandler:

If Not oConnection3 Is Nothing Then
    If oConnection3.State = adStateOpen Then oConnection3.Close
    Set oConnection3 = Nothing
End If
If Not m_oRecordset Is Nothing Then
    If m_oRecordset.State = adStateOpen Then m_oRecordset.Close
    Set m_oRecordset = Nothing
End If
End Sub

'EndHelloData

```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

OLE DB Providers

The ADO Programmer's Guide [Introduction](#) discusses the relationship between ADO and the rest of the Microsoft Data Access architecture. OLE DB defines a set of COM interfaces to provide applications with uniform access to data that is stored in diverse information sources. This approach allows a data source to share its data through the interfaces that support the amount of DBMS functionality appropriate to the data source. By design, the high-performance architecture of OLE DB is based on its use of a flexible, component-based services model. Rather than having a prescribed number of intermediary layers between the application and the data, OLE DB requires only as many components as are needed to accomplish a particular task.

For example, suppose a user wants to run a query. Consider the following scenarios:

- The data resides in a relational database for which there currently exists an ODBC driver but no native OLE DB provider: The application uses ADO to talk to the OLE DB Provider for ODBC, which then loads the appropriate ODBC driver. The driver passes the SQL statement to the DBMS, which retrieves the data.
- The data resides in Microsoft SQL Server for which there is a native OLE DB provider: The application uses ADO to talk directly to the OLE DB Provider for Microsoft SQL Server. No intermediaries are required.
- The data resides in Microsoft Exchange Server, for which there is an OLE DB provider but which does not expose an engine to process SQL queries: The application uses ADO to talk to the OLE DB Provider for Microsoft Exchange and calls upon an OLE DB query processor component to handle the querying.
- The data resides in the Microsoft NTFS file system in the form of documents: Data is accessed by using a native OLE DB provider over Microsoft Indexing Service, which indexes the content and properties of documents in the file system to enable efficient content searches.

In all of the preceding examples, the application can query the data. The user's needs are met with a minimum number of components. In each case, additional components are used only if needed, and only the required components are

invoked. This demand-loading of reusable and shareable components greatly contributes to high performance when OLE DB is used.

Providers fall into two categories: those providing data and those providing services. A [data provider](#) owns its own data and exposes it in tabular form to your application. A [service provider](#) encapsulates a service by producing and consuming data, augmenting features in your ADO applications. A service provider may also be further defined as a [service component](#), which must work in conjunction with other service providers or components.

ADO provides a consistent, higher level interface to the various OLE DB providers.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Data Providers

Data providers represent diverse sources of data such as SQL databases, indexed-sequential files, spreadsheets, document stores, and mail files. Providers expose data uniformly using a common abstraction called the rowset.

ADO is powerful and flexible because it can connect to any of several different data providers and still expose the same programming model, regardless of the specific features of any given provider. However, because each data provider is unique, how your application interacts with ADO will vary by data provider.

For example, the capabilities and features of the OLE DB Provider for SQL Server, which is used to access Microsoft SQL Server databases, are considerably different from those of the Microsoft OLE DB Provider for Internet Publishing, which is used to access file stores on a Web server.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Service Providers and Components

Service providers are components that extend the functionality of data providers by implementing extended interfaces that are not natively supported by the data store.

Microsoft Data Access provides a *component architecture* that allows individual, specialized components to implement discrete sets of database functionality, or "services," on top of less capable stores. Thus, rather than forcing each data store to provide its own implementation of extended functionality or forcing generic applications to implement database functionality internally, service components provide a common implementation that any application can use when accessing any data store. The fact that some functionality is implemented natively by the data store and some through generic components is transparent to the application.

For example, a cursor engine, such as the Microsoft Cursor Service for OLE DB, is a service component that can consume data from a sequential, forward-only data store to produce scrollable data. Other service providers commonly used by ADO include the Microsoft OLE DB Persistence Provider (for saving data to a file), the Microsoft Data Shaping Service for OLE DB (for hierarchical **Recordsets**), and the Microsoft OLE DB Remoting Provider (for invoking data providers on a remote computer).

For more information about service and data providers, see [Appendix A: Providers](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Errors

Any operation involving ADO objects can generate one or more provider errors. As each error occurs, one or more **Error** objects are placed in the **Errors** collection of the **Connection** object. For details about handling warnings and errors in your ADO application, see [Chapter 6: Error Handling](#).

Application errors can be raised by a separate mechanism. For example, in Visual Basic, the **Err** object will contain application-level errors.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Chapter 2: Getting Data

The preceding chapter introduced four primary operations involved in creating an ADO application: getting data, examining data, editing data, and updating data. This chapter will focus on the details of the concepts relevant to the first operation: getting data.

Several ADO objects can play a role in this operation. First you connect to the data source using an ADO **Connection** object (which at times will be created implicitly). Then you pass instructions to the data source about what you want to do using an ADO **Command** object (which also can be created implicitly). The result of passing a command to a data source and receiving its response usually will be represented in an ADO **Recordset** object.

To get data, your application must be in communication with a data source, such as a DBMS, a file store, or a comma-delimited text file. This communication represents a *connection*—the environment necessary for exchanging data.

The ADO object model represents the concept of a connection with the **Connection** object—the foundation upon which much ADO functionality is built. The purpose of a **Connection** object is to:

- Define the information ADO needs to communicate with data sources and create sessions.
- Define the transactional capabilities of the session.
- Allow you to create and execute commands against the data source.
- Provide information about the design of the underlying data source in the form of schema rowsets. For more information about schema rowsets, see [OpenSchema Method](#).

ADO 2.5 

Making a Connection

To connect to a data source, you must specify a *connection string*, the parameters of which might differ for each provider and data source. For more information, see [Creating the Connection String](#).

ADO most commonly opens a connection by using the **Connection** object **Open** method. The syntax for the **Open** method is shown here:

```
Dim connection as New ADODB.Connection  
connection.Open ConnectionString, UserID, Password, OpenOptions
```

Alternatively, you can invoke a shortcut technique, **Recordset.Open**, to open an implicit connection and issue a command over that connection in one operation. Do this by passing in a valid connection string as the *ActiveConnection* argument to the **Open** method. Here is the syntax for each method in Visual Basic:

```
Dim recordset as ADODB.Recordset  
Set recordset = New ADODB.Recordset  
recordset.Open Source, ActiveConnection, CursorType, LockType, Optio
```

Note When should you use a **Connection** object vs. the **Recordset.Open** shortcut? Use the **Connection** object if you plan to open more than one **Recordset**, or when executing multiple commands. A connection is still created by ADO implicitly when you use the **Recordset.Open** shortcut.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Using the Connection Object

A **Connection** object represents a unique session with a data source. In the case of a client/server database system, it can be equivalent to an actual network connection to the server. Depending on the functionality supported by the provider, some collections, methods, or properties of a **Connection** object might not be available.

Before opening a **Connection** object, you must define certain information about the data source and type of connection. The *ConnectionString* parameter of the **Connection** object **Open** method—or the **ConnectionString** property on the **Connection** object—usually contains most of this information. A connection string is a string of characters that defines a variable number of arguments. The arguments—some required by ADO, but others provider-specific—contain information that the **Connection** object must have to carry out its work. The arguments that make up the *ConnectionString* parameter are separated with semicolons (;).

Note You can also specify an ODBC Data Source Name (DSN) or a Data Link (UDL) file in a connection string. For more information about DSNs, see Data Sources in Part 1 of the *ODBC Programmer's Reference*. For more information about UDLs, see Data Link API Overview in the *OLE DB Programmer's Reference*.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Creating the Connection String

ADO directly supports five arguments in a connection string. Other arguments are passed to the provider that is named in the *Provider* argument without any processing by ADO.

Argument	Description
<i>Provider</i>	Specifies the name of a provider to use for the connection.
<i>File Name</i>	Specifies the name of a provider-specific file (for example, a persisted data source object) containing preset connection information.
<i>URL</i>	Specifies the connection string as an absolute URL identifying a resource, such as a file or directory.
<i>Remote Provider</i>	Specifies the name of a provider to use when opening a client-side connection. (Remote Data Service only.)
<i>Remote Server</i>	Specifies the path name of the server to use when opening a client-side connection. (Remote Data Service only.)

Note In the following examples and throughout the ADO Programmer's Guide, the user id "MyId" with a password of "123aBc" is used to authenticate against the server. You should substitute these values with valid login credentials for your server. Also, substitute the name of your server for "MySqlServer".

The HelloData application in Chapter 1 used the following connection string:

```
m_sConnStr = "Provider='SQLOLEDB';Data Source='MySqlServer';" & _  
            "Initial Catalog='Northwind';Integrated Security='SSPI'
```

The only ADO parameter supplied in this connection string was

"Provider=SQLOLEDB", which indicated the Microsoft OLE DB Provider for SQL Server. Other valid parameters that can be passed in the connection string can be determined by referring to individual providers' documentation. According to the OLE DB Provider for SQL Server documentation, you can substitute "Server" for the *Data Source* parameter and "Database" for the *Initial Catalog* parameter. Thus, the following connection string would produce results identical to the first:

```
m_sConnStr = "Provider='SQLOLEDB';Server='MySqlServer';" & _  
            "Database='Northwind';Integrated Security='SSPI';"
```

To open the connection, simply pass the connection string as the first argument in the **Connection** object **Open** method:

```
objConn.Open m_sConnStr
```

It is also possible to supply much of this information by setting properties of the **Connection** object before opening the connection. For example, you could achieve the same effect as the connection string above by using the following code:

```
With objConn  
    .Provider = "SQLOLEDB"  
    .DefaultDatabase = "Northwind"  
    .Properties("Data Source") = "MySqlServer"  
    .Properties("Integrated Security") = "SSPI"  
    .Open  
End With
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Controlling Transactions

A *transaction* delimits the beginning and end of a series of data access operations that transpire across a connection. Subject to the transactional capabilities of your data source, the **Connection** object also allows you to create and manage transactions. For example, using the Microsoft OLE DB Provider for SQL Server to access a database on Microsoft SQL Server 2000, you can create multiple nested transactions for the commands you execute.

ADO ensures that changes to a data source resulting from operations in a transaction occur successfully together or not at all.

If you cancel the transaction, or if one of its operations fails, the ultimate result will be as if none of the operations in the transaction occurred. The data source will remain as it was before the transaction began.

The ADO object model does not explicitly include transactions, but represents them with a set of **Connection** object methods (**BeginTrans**, **CommitTrans**, and **RollbackTrans**).

For more information about transactions, see [Chapter 5: Updating and Persisting Data](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Using the Command Object

After connecting to a data source, you need to execute requests against it to obtain result sets. ADO encapsulates this type of command functionality in the **Command** object.

You can use the **Command** object to request any type of operation from the provider, assuming that the provider can interpret the command string properly. A common operation for data providers is to query a database and return records in a **Recordset** object. **Recordsets** will be discussed later in this and other chapters; for now, think of them as tools to hold and view result sets. As with many ADO objects, depending on the functionality of the provider, some **Command** object collections, methods, or properties might generate errors when referenced.

It is not always necessary to create a **Command** object to execute a command against a data source. You can use the **Execute** method on the **Connection** object or the **Open** method on the **Recordset** object. However, you should use a **Command** object if you need to reuse a command in your code or if you need to pass detailed parameter information with your command. These scenarios are covered in more detail later in this chapter.

Note Certain **Commands** can return a result set as a binary stream or as a single **Record** rather than as a **Recordset**, if this is supported by the provider. Also, some **Commands** are not intended to return any result set at all (for example, a SQL Update query). This chapter will cover the most typical scenario, however: executing **Commands** that return results into a **Recordset** object. For more information about returning results into **Records** or **Streams**, see [Chapter 10: Records and Streams](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Command Object Overview

With the collections, methods, and properties of a **Command** object, you can do the following:

- Define the executable text of the command (for example, a SQL statement or a stored procedure) by using the **CommandText** property.
- Define parameterized queries or stored procedure arguments by using **Parameter** objects and the **Parameters** collection.
- Execute a command and return a **Recordset** object, if appropriate, by using the **Execute** method.
- Specify the type of command by using the **CommandType** property prior to execution to optimize performance.
- Control whether the provider saves a prepared (or compiled) version of the command prior to execution by using the **Prepared** property.
- Set the number of seconds that a provider will wait for a command to execute by using the **CommandTimeout** property.
- Associate an open connection with a **Command** object by setting its **ActiveConnection** property.
- Set the **Name** property to identify the **Command** object as a method on the associated **Connection** object.
- Pass a **Command** object to the **Source** property of a **Recordset** in order to obtain data.

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Creating and Executing a Simple Command

Though not a typical usage of the **Command** object, the following code shows the basic method of using the **Command** object to execute a command against a data source. In this case, it is a row-returning command, so it returns the results of the command execution into a **Recordset** object.

```
'BeginBasicCmd
On Error GoTo ErrHandler:

Dim objConn As New ADODB.Connection
Dim objCmd As New ADODB.Command
Dim objRs As New ADODB.Recordset

objCmd.CommandText = "SELECT OrderID, OrderDate, " & _
                    "RequiredDate, ShippedDate " & _
                    "FROM Orders " & _
                    "WHERE CustomerID = 'ALFKI' " & _
                    "ORDER BY OrderID"

objCmd.CommandType = adCmdText

' Connect to the data source.
Set objConn = GetNewConnection
objCmd.ActiveConnection = objConn

' Execute once and display...
Set objRs = objCmd.Execute

Debug.Print "ALFKI"
Do While Not objRs.EOF
    Debug.Print vbTab & objRs(0) & vbTab & objRs(1) & vbTab & _
                objRs(2) & vbTab & objRs(3)
    objRs.MoveNext
Loop

'clean up
objRs.Close
objConn.Close
Set objRs = Nothing
Set objConn = Nothing
Set objCmd = Nothing
Exit Sub
```

```

ErrorHandler:
    'clean up
    If objRs.State = adStateOpen Then
        objRs.Close
    End If

    If objConn.State = adStateOpen Then
        objConn.Close
    End If

    Set objRs = Nothing
    Set objConn = Nothing
    Set objCmd = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
'EndBasicCmd

```

The command to be executed is specified with the **CommandText** property.

Note Several examples in this section call a utility function, `GetNewConnection`, to establish a connection with the data provider. To avoid redundancy, it is listed only once, here:

```

'BeginNewConnection
Private Function GetNewConnection() As ADODB.Connection
    Dim oCn As New ADODB.Connection
    Dim sCnStr As String

    sCnStr = "Provider='SQLOLEDB';Data Source='MySqlServer';" & _
            "Integrated Security='SSPI';Database='Northwind';"
    oCn.Open sCnStr

    If oCn.State = adStateOpen Then
        Set GetNewConnection = oCn
    End If

End Function
'EndNewConnection

```

ADO 2.5 

Command Object Parameters

A more interesting use for the **Command** object is shown in the next example, in which the text of the SQL command has been modified to make it parameterized. This makes it possible to reuse the command, passing in a different value for the parameter each time. Because the **Prepared** property on the **Command** object is set equal to **True**, ADO will require the provider to compile the command specified in **CommandText** before executing it for the first time. It also will retain the compiled command in memory. This slows the execution of the command slightly the first time it is executed because of the overhead required to prepare it, but results in a performance gain each time the command is called thereafter. Thus, commands should be prepared only if they will be used more than once.

```
'BeginManualParamCmd
  On Error GoTo ErrHandler:

  Dim objConn As New ADODB.Connection
  Dim objCmd As New ADODB.Command
  Dim objParm1 As New ADODB.Parameter
  Dim objRs As New ADODB.Recordset

  ' Set the CommandText as a parameterized SQL query.
  objCmd.CommandText = "SELECT OrderID, OrderDate, " & _
    "RequiredDate, ShippedDate " & _
    "FROM Orders " & _
    "WHERE CustomerID = ? " & _
    "ORDER BY OrderID"
  objCmd.CommandType = adCmdText

  ' Prepare command since we will be executing it more than once.
  objCmd.Prepared = True

  ' Create new parameter for CustomerID. Initial value is ALFKI.
  Set objParm1 = objCmd.CreateParameter("CustId", adChar, _
    adParamInput, 5, "ALFKI")
  objCmd.Parameters.Append objParm1

  ' Connect to the data source.
  Set objConn = GetNewConnection
  objCmd.ActiveConnection = objConn

  ' Execute once and display...
```

```

Set objRs = objCmd.Execute

Debug.Print objParm1.Value
Do While Not objRs.EOF
    Debug.Print vbTab & objRs(0) & vbTab & objRs(1) & vbTab & _
                objRs(2) & vbTab & objRs(3)
    objRs.MoveNext
Loop

' ...then set new param value, re-execute command, and display.
objCmd("CustId") = "CACTU"
Set objRs = objCmd.Execute

Debug.Print objParm1.Value
Do While Not objRs.EOF
    Debug.Print vbTab & objRs(0) & vbTab & objRs(1) & vbTab & _
                objRs(2) & vbTab & objRs(3)
    objRs.MoveNext
Loop

'clean up
objRs.Close
objConn.Close
Set objRs = Nothing
Set objConn = Nothing
Set objCmd = Nothing
Set objParm1 = Nothing
Exit Sub

ErrorHandler:
'clean up
If objRs.State = adStateOpen Then
    objRs.Close
End If

If objConn.State = adStateOpen Then
    objConn.Close
End If

Set objRs = Nothing
Set objConn = Nothing
Set objCmd = Nothing
Set objParm1 = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
'EndManualParamCmd

```

Not all providers support prepared commands. If the provider does not support command preparation, it might return an error as soon as this property is set to **True**. If it does not return an error, it ignores the request to prepare the command and sets the **Prepared** property to **False**.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Calling a Stored Procedure with a Command

You can also use a command when calling a stored procedure. The following code calls a stored procedure in the Northwind sample database, called `CustOrdersOrders`, which is defined as follows:

```
CREATE PROCEDURE CustOrdersOrders @CustomerID nchar(5) AS
SELECT OrderID, OrderDate, RequiredDate, ShippedDate
FROM Orders
WHERE CustomerID = @CustomerID
ORDER BY OrderID
```

This stored procedure is similar to the command used in [Command Object Parameters](#), in that it takes a customer ID parameter and returns information about that customer's orders. The code below uses this stored procedure as the source for an ADO **Recordset**.

Using the stored procedure allows you to access another capability of ADO: the **Parameters** collection **Refresh** method. By using this method, ADO can automatically fill in all information about the parameters required by the command at run time. There is a performance penalty in using this technique, because ADO must query the data source for the information about the parameters.

Other important differences exist between the code below and the code in [Command Object Parameters](#), where the parameters were entered manually. First, this code does not set the **Prepared** property to **True** because it is a SQL Server stored procedure and is precompiled by definition. Second, the **CommandType** property of the **Command** object changed to **adCmdStoredProc** in the second example to inform ADO that the command was a stored procedure.

```
'BeginAutoParamCmd
  On Error GoTo ErrHandler:

  Dim objConn As New ADODB.Connection
  Dim objCmd As New ADODB.Command
```

```

Dim objParm1 As New ADODB.Parameter
Dim objRs As New ADODB.Recordset

' Set CommandText equal to the stored procedure name.
objCmd.CommandText = "CustOrdersOrders"
objCmd.CommandType = adCmdStoredProc

' Connect to the data source.
Set objConn = GetNewConnection
objCmd.ActiveConnection = objConn

' Automatically fill in parameter info from stored procedure.
objCmd.Parameters.Refresh

' Set the param value.
objCmd(1) = "ALFKI"

' Execute once and display...
Set objRs = objCmd.Execute

Debug.Print objParm1.Value
Do While Not objRs.EOF
    Debug.Print vbTab & objRs(0) & vbTab & objRs(1) & vbTab & _
                objRs(2) & vbTab & objRs(3)
    objRs.MoveNext
Loop

' ...then set new param value, re-execute command, and display.
objCmd(1) = "CACTU"
Set objRs = objCmd.Execute

Debug.Print objParm1.Value
Do While Not objRs.EOF
    Debug.Print vbTab & objRs(0) & vbTab & objRs(1) & vbTab & _
                objRs(2) & vbTab & objRs(3)
    objRs.MoveNext
Loop

'clean up
objRs.Close
objConn.Close
Set objRs = Nothing
Set objConn = Nothing
Set objCmd = Nothing
Set objParm1 = Nothing
Exit Sub

```

ErrorHandler:

```

'clean up
If objRs.State = adStateOpen Then

```

```
        objRs.Close
    End If

    If objConn.State = adStateOpen Then
        objConn.Close
    End If

    Set objRs = Nothing
    Set objConn = Nothing
    Set objCmd = Nothing
    Set objParm1 = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
'EndAutoParamCmd
```

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Named Commands

You can set the **Name** property on a **Command** object and then execute the command by calling it as if it were a method on the **Command** object **ActiveConnection** property. This is illustrated in the following example, in which the command is named *GetCustomers*. Notice that the code passes in a declared and instantiated **Recordset** object to the *GetCustomers* "method." You can also pass in parameters to the "method" if they are required by the **Command**.

```
'BeginNamedCmd
  On Error GoTo ErrHandler:

  Dim objConn As New ADODB.Connection
  Dim objCmd As New ADODB.Command
  Dim objRs As New ADODB.Recordset

  ' Connect to the data source.
  Set objConn = GetNewConnection

  objCmd.CommandText = "SELECT CustomerID, CompanyName FROM Custom
  objCmd.CommandType = adCmdText

  'Name the command.
  objCmd.Name = "GetCustomers"

  objCmd.ActiveConnection = objConn

  ' Execute using Command.Name from the Connection.
  objConn.GetCustomers objRs

  ' Display.
  Do While Not objRs.EOF
    Debug.Print objRs(0) & vbTab & objRs(1)
    objRs.MoveNext
  Loop

  'clean up
  objRs.Close
  objConn.Close
  Set objRs = Nothing
  Set objConn = Nothing
  Set objCmd = Nothing
  Exit Sub
```

```
ErrorHandler:
    'clean up
    If objRs.State = adStateOpen Then
        objRs.Close
    End If

    If objConn.State = adStateOpen Then
        objConn.Close
    End If

    Set objRs = Nothing
    Set objConn = Nothing
    Set objCmd = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
'EndNamedCmd
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Adding Data to a Recordset

The **Recordset** is probably the most used of the ADO objects. In ADO a **Recordset** is best thought of as the combination of a result set from a data source and its associated cursor behaviors. Thus, you can put data into a **Recordset** and then use the **Recordset** methods and properties to navigate through the rows of data, view the values in the rows, and otherwise manipulate the result set.

This section will focus on adding data to the **Recordset**. For information about navigating through the data or updating the data, see [Chapter 4: Editing Data](#) and [Chapter 5: Updating and Persisting Data](#). You do not always need the advanced capabilities of a **Command** object to add your result set to a **Recordset**. Often, you can execute your command by setting the **Source** property on the **Recordset** or passing a command string to the **Recordset** object **Open** method.

There are a variety of ways to add data from your data source to a **Recordset**. The technique you use depends on the needs of your application and the capabilities of your provider.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

The Recordset Object Open Method

Everything you need to open an ADO **Recordset** is built into the **Open** method. You can use it without explicitly creating any other objects. The syntax of this method is as follows:

recordset.**Open** *Source, ActiveConnection, CursorType, LockType, Optio*

All arguments are optional because the information they pass can be communicated to ADO in other ways. However, understanding each argument will help you to understand many important ADO concepts. The following topics will examine each argument of this method in more detail.

Source and Options Arguments

The *Source* and *Options* arguments appear in the same topic because they are closely related.

`recordset.Open` **Source**, *ActiveConnection*, *CursorType*, *LockType*, **Optio**

The *Source* argument is a **VARIANT** that evaluates to a valid **Command** object, a text command (e.g., a SQL statement), a table name, a stored procedure call, a URL, or the name of a file or **Stream** object containing a persistently stored **Recordset**. If *Source* is a file path name, it can be a full path ("C:\dir\file.rst"), a relative path ("..\file.rst"), or a URL ("http://files/file.rst"). You can also specify this information in the **Recordset** object **Source** property and leave the *Source* argument blank.

The *Options* argument is a **Long** value that indicates either or both of the following:

- How the provider should evaluate the *Source* argument if it represents something other than a **Command** object.
- That the **Recordset** should be restored from a file where it was previously saved.

This argument can contain a bitmask of **CommandTypeEnum** or **ExecuteOptionEnum** values. A **CommandTypeEnum** passed in the *Options* argument sets the **CommandType** property of the **Recordset**.

Note The **ExecuteOpenEnum** values of **adExecuteNoRecords** and **adExecuteStream** cannot be used with **Open**.

If the **CommandType** property value equals **adCmdUnknown** (the default value), you might experience diminished performance, because ADO must make calls to the provider to determine whether the **CommandText** property is a SQL statement, a stored procedure, or a table name. If you know what type of command you are using, setting the **CommandType** property instructs ADO to go directly to the relevant code. If the **CommandType** property does not match the type of command in the **CommandText** property, an error occurs when you call the **Open** method.

For more information about using these enumerated constants for *Options* and with other ADO methods and properties, see [CommandTypeEnum](#) and [ExecuteOptionEnum](#).

ActiveConnection Argument

You can pass in either a **Connection** object or a connection string as the *ActiveConnection* argument.

*recordset.Open Source, **ActiveConnection**, CursorType, LockType, Optio*

The *ActiveConnection* argument corresponds to the **ActiveConnection** property and specifies in which connection to open the **Recordset** object. If you pass a connection definition for this argument, ADO opens a new connection using the specified parameters. After opening the **Recordset** with a client-side cursor (**CursorLocation** = **adUseClient**), you can change the value of this property to send updates to another provider. Or you can set this property to Nothing (in Microsoft Visual Basic) or NULL to disconnect the **Recordset** from any provider. Changing **ActiveConnection** for a server-side cursor generates an error, however.

If you pass a **Command** object in the *Source* argument and also pass an *ActiveConnection* argument, an error occurs because the **ActiveConnection** property of the **Command** object must already be set to a valid **Connection** object or connection string.

CursorType Argument

*recordset.Open Source, ActiveConnection, **CursorType**, LockType, Optio*

As discussed in [The Significance of Cursor Location](#), the type of cursor that your application uses will determine which capabilities are available to the resultant **Recordset** (if any). For a detailed examination of cursor types, see [Chapter 8: Understanding Cursors and Locks](#).

The *CursorType* argument can accept any of the **CursorTypeEnum** values.

LockType Argument

*recordset.Open Source, ActiveConnection, CursorType, **LockType**, Optio*

Set the *LockType* argument to specify what type of locking the provider should use when opening the **Recordset**. The different types of locking are discussed in [Chapter 8: Understanding Cursors and Locks](#).

The *LockType* argument can accept any of the **LockTypeEnum** values.

Retrieving Multiple Recordsets

You might occasionally need to execute a command that will return more than one result set. A common example is a stored procedure that runs against a SQL Server database, as in the following example. The stored procedure contains a COMPUTE clause to return the average price of all products in the table. The definition of the stored procedure is as follows:

```
CREATE PROCEDURE ProductsWithAvgPrice
AS
SELECT ProductID, ProductName, UnitPrice
FROM PRODUCTS
COMPUTE AVG(UnitPrice)
```

The [Microsoft OLE DB Provider for SQL Server](#) returns multiple result sets to ADO when the command contains a COMPUTE clause. Therefore, the ADO code must use the **NextRecordset** method to access the data in the second result set, as shown here:

```
'BeginNextRs
  On Error GoTo ErrHandler:

  Dim objConn As New ADODB.Connection
  Dim objCmd As New ADODB.Command
  Dim objRs As New ADODB.Recordset

  Set objConn = GetNewConnection
  objCmd.ActiveConnection = objConn

  objCmd.CommandText = "ProductsWithAvgPrice"
  objCmd.CommandType = adCmdStoredProc

  Set objRs = objCmd.Execute

  Do While Not objRs.EOF
    Debug.Print objRs(0) & vbTab & objRs(1) & vbTab & _
               objRs(2)
    objRs.MoveNext
  Loop

  Set objRs = objRs.NextRecordset

  Debug.Print "AVG. PRICE = $ " & objRs(0)
```

```
'clean up
objRs.Close
objConn.Close
Set objRs = Nothing
Set objConn = Nothing
Set objCmd = Nothing
Exit Sub
```

ErrorHandler:

```
'clean up
If objRs.State = adStateOpen Then
    objRs.Close
End If

If objConn.State = adStateOpen Then
    objConn.Close
End If

Set objRs = Nothing
Set objConn = Nothing
Set objCmd = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
'EndNextRs
```

For more information, see [NextRecordset](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Chapter 3: Examining Data

Chapter 2 explained how to retrieve data from a data source as a **Recordset** object. This chapter will discuss the **Recordset** in more detail, including how to navigate through the **Recordset** and view its data.

The following diagram illustrates the object model of the **Recordset** object. Click an object or collection for more information.



Recordsets have methods and properties designed to make it easy to move through them and examine their contents. Depending on the functionality supported by the provider, some **Recordset** methods or properties might not be available. To continue exploring the **Recordset** object, consider a **Recordset** that would be returned from the Northwind sample database on Microsoft SQL Server 2000, using the following code:

```
'BeginRsTour
Public Sub RecordsetTour()
    On Error GoTo ErrHandler:

    Dim objRs As New ADODB.Recordset
    Dim strSQL As String

    strSQL = "SELECT ProductID, ProductName, UnitPrice FROM Products
            "WHERE CategoryID = 7"                '7 = Produce

    objRs.Open strSQL, strConnStr, adOpenForwardOnly, _
        adLockReadOnly, adCmdText

    'Clean up
    objRs.Close
    Set objRs = Nothing
    Exit Sub

ErrHandler:
    If Not objRs Is Nothing Then
        If objRs.State = adStateOpen Then objRs.Close
```

```

        Set objRs = Nothing
    End If

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndRsTour

```

This SQL query returns a **Recordset** with five rows (records) and three columns (fields). The values for each row are shown in the following table.

FIELD 0 Name = ProductID	FIELD 1 Name = ProductName	FIELD 2 Name = UnitPrice
7	Uncle Bob's Organic Dried Pears	30.0000
14	Tofu	23.2500
28	Rssle Sauerkraut	45.6000
51	Manjimup Dried Apples	53.0000
74	Longlife Tofu	10.0000

The next section will explain how to locate the current position of the cursor in this sample **Recordset**.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Locating the Current Record

The current position of the cursor in the **Recordset** delineates the current record position. Assuming that the command issued returns results, the cursor is automatically placed at the first record when the **Recordset Open** method is called. So, with the sample **Recordset**, the cursor would be on the first record, "Uncle Bob's Organic Dried Pears."

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Counting Rows

The **RecordCount** property returns a **Long** value that indicates the number of records in the **Recordset**. Use the **RecordCount** property to find out how many records are in a **Recordset** object. The property returns -1 when ADO cannot determine the number of records or if the provider or cursor type does not support **RecordCount**. Reading the **RecordCount** property on a closed **Recordset** causes an error.

The **RecordCount** property depends on the capabilities of the provider and the type of cursor. The **RecordCount** property will return -1 for a forward-only cursor, the actual count for a static or keyset cursor, and either -1 or the actual count for a dynamic cursor, depending on the data source.

The sample **Recordset** introduced in [Examining Data](#) would return -1 because a forward-only cursor was opened. In order to use the **RecordCount** property, you would need to open the **Recordset** with a more sophisticated cursor (static or keyset).

In certain cases, your provider or cursor might be unable to provide the **RecordCount** value without first fetching all records from the data source. To force this type of fetch, call the **Recordset MoveLast** method before calling **RecordCount**.

If you were to replace the line of code that calls the **Recordset Open** method with the following:

```
oRs.Open sSQL, sCnStr, adOpenStatic, adLockOptimistic, adCmdText
```

you would be able to use the **RecordCount** property because static cursors with the [Microsoft OLE DB Provider for SQL Server](#) support **RecordCount**. For example, the following code would print out the number of records returned by the command to the debug window, assuming the cursor supports the **RecordCount** property:

```
Debug.Print oRs.RecordCount ' Output: 4
```

From this point forward, assume that these more capable (but more expensive)

cursor and lock type settings are used.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

The Limits of a Recordset

Use the **BOF** and **EOF** properties to determine whether a **Recordset** object contains records or whether you've gone beyond the limits of a **Recordset** object when you move from record to record. Think of **BOF** and **EOF** as "phantom" records that are positioned at the beginning and end of the **Recordset**. Building on the sample **Recordset** from [Examining Data](#), it would now look like this:

ProductID	ProductName	UnitPrice
BOF		
7	Uncle Bob's Organic Dried Pears	30.0000
14	Tofu	23.2500
28	Rssle Sauerkraut	45.6000
51	Manjimup Dried Apples	53.0000
74	Longlife Tofu	10.0000
EOF		

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

If either the **BOF** or **EOF** property is **True**, there is no current record, as shown in the following code:

```
If oRs.BOF And oRs.EOF Then
    ' Command returned no records.
End If
```

If you open a **Recordset** object containing no records, the **BOF** and **EOF** properties are both set to **True** and the value of the **Recordset** object's **RecordCount** property setting depends on the cursor type. -1 will be returned for dynamic cursors (**CursorType** = **adOpenDynamic**) and 0 will be returned

for other cursors.

When you open a **Recordset** object that contains at least one record, the first record is the current record and the **BOF** and **EOF** properties are **False**.

If you delete the last remaining record in the **Recordset** object, the cursor is left in an indeterminate state. The **BOF** and **EOF** properties may remain **False** until you attempt to reposition the current record, depending upon the provider.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Navigating Through the Data

Now that you have executed a command against the data source and determined that the result set contains data, you can move through the results by using the navigation methods and properties provided by the **Recordset** object. The following topics describe how to use these methods and properties on the sample **Recordset**:

- [Jumping to a Record](#)
- [More Ways to Move in a Recordset](#)
- [Using Bookmarks](#)
- [Using Pages](#)
- [Recordset Positioning](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Jumping to a Record

The [Move](#) method allows you to move forward or backward in the **Recordset** a specified number of records by using the following syntax:

```
oRs.Move NumRecords, Start
```

The **Move** method is supported on all **Recordset** objects.

If the *NumRecords* argument is greater than zero, the current record position moves forward (toward the end of the **Recordset**). If *NumRecords* is less than zero, the current record position moves backward (toward the beginning of the **Recordset**).

If the **Move** call would move the current record position to a point before the first record, ADO sets the current record to the position before the first record in the **Recordset** (**BOF** is **True**). An attempt to move backward when the **BOF** property is already **True** generates an error.

If the **Move** call would move the current record position to a point after the last record, ADO sets the current record to the position after the last record in the **Recordset** (**EOF** is **True**). An attempt to move forward when the **EOF** property is already **True** generates an error.

Calling the **Move** method from an empty **Recordset** object generates an error.

If you pass a bookmark in the *Start* argument, the move is relative to the record with this bookmark, assuming the **Recordset** object supports bookmarks. A bookmark is obtained by using the [Bookmark](#) property. If not specified, the move is relative to the current record.

If you are using the **CacheSize** property to locally cache records from the provider, passing a *NumRecords* argument that moves the current record position outside the current group of cached records forces ADO to retrieve a new group of records, starting from the destination record. The **CacheSize** property determines the size of the newly retrieved group, and the destination record is the first record retrieved.

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

More Ways to Move in a Recordset

The following four methods are used to move around, or scroll, in the **Recordset**: [MoveFirst](#), [MoveLast](#), [MoveNext](#), and [MovePrevious](#). (Some of these methods are unavailable on forward-only cursors.)

MoveFirst changes the current record position to the first record in the **Recordset**. **MoveLast** changes the current record position to the last record in the **Recordset**. To use **MoveFirst** or **MoveLast**, the **Recordset** object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error.

MoveNext moves the current record position one place forward. If you are on the last record when you call **MoveNext**, **EOF** will be set to **True**.

MovePrevious moves the current record position one place backward. If you are on the first record when you call **MovePrevious**, **BOF** will be set to **True**. It is wise to check the **EOF** and **BOF** properties when using these methods and to move the cursor back to a valid current record position if you move off either end of the **Recordset**, as shown here:

```
. . .  
oRs.MoveNext  
If oRs.EOF Then oRs.MoveLast  
. . .
```

Or, in the case of the **MovePrevious** method:

```
. . .  
oRs.MovePrevious  
If oRs.BOF Then oRs.MoveFirst  
. . .
```

In cases where the **Recordset** has been filtered or sorted and the current record's data is changed, the position may also change. In such cases the **MoveNext** method works normally, but be aware that the position is moved one record forward from the new position, not the old position. For example, changing the data in the current record, such that the record is moved to the end of the sorted **Recordset**, would mean that calling **MoveNext** results in ADO setting the current record to the position after the last record in the **Recordset** (**EOF** =

True).

The behavior of the various Move methods of the **Recordset** object depends, to some extent, on the data within the **Recordset**. New records added to the **Recordset** are initially added in a particular order, which is defined by the data source and may be dependent implicitly or explicitly on the data in the new record. For example, if a sort or a join is done within the query that populates the **Recordset**, the new record will be inserted in the appropriate place within the **Recordset**. If ordering is not explicitly specified when creating the **Recordset**, changes in the data source implementation may cause the ordering of the returned rows to change inadvertently. In addition, the sorting, filtering, and editing functions of the **Recordset** can affect the order and possibly which rows in the recordset will be visible.

Therefore, **MoveNext**, **MovePrevious**, **MoveFirst**, **MoveLast**, and **Move** are all sensitive to other operations performed on the same **Recordset**. ADO will always try to maintain your current position until you explicitly move it, but sometimes intervening changes make it difficult to understand the effects of a subsequent move. For example, if you call **MoveFirst** to position on the first row of a sorted **Recordset** and you change the sort from ascending order to descending order, you are still on the same row—but now it is the last row in the **Recordset**. **MoveFirst** will take you to a different row (the new first row).

As another example, if you are positioned on a particular row in the middle of a **Recordset** and you call **Delete** and then call **MoveNext**, you are now on the record immediately following the deleted record. But calling **MovePrevious** makes the record preceding the one you deleted the current record, because the deleted record is no longer counted in the active membership of the **Recordset**.

It is particularly difficult to define consistent move semantics across all providers for methods that move relative to the current record—**MovePrevious**, **MoveNext**, and **Move**—in the face of changing data in the current record. For example, if you are working with a sorted, filtered **Recordset**, and you change the data in the current record so that it would precede all other records, but your changed data also no longer matches the filter, it is not clear where a **MoveNext** operation should take you. The safest conclusion is that relative movement within a **Recordset** is riskier than absolute movement (such as using **MoveFirst** or **MoveLast**) when the data is changing while records are being edited, added, or deleted. Sorting and filtering should be based on a primary key or ID, because

this type of value should not change.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Using Bookmarks

It is often useful to return directly to a specific record after having moved around in the **Recordset** without having to scroll through every record and compare values. For example, if you attempt to search for a record using the **Find** method but the search returns no records, you are automatically placed at either end of the **Recordset**. If your provider supports them, bookmarks can be used to mark your place before using the **Find** method so you can return to your location. A bookmark is a **Variant** type value that uniquely identifies a record in a **Recordset** object.

You can also use a variant array of bookmarks with the **Recordset Filter** method to filter on a selected set of records. For details about this technique, see Filtering the Results in the topic, [Working with Recordsets](#), later in this chapter.

You can use the **Bookmark** property to get a bookmark for a record, or set the current record in a **Recordset** object to the record identified by a valid bookmark. The following code uses the **Bookmark** property to set a bookmark and then return to the bookmarked record after moving on to other records. To determine if your **Recordset** supports bookmarks, use the **Supports** method.

```
'BeginBookmarkEg
  Dim varBookmark As Variant
  Dim blnCanBkmrk As Boolean

  objRs.Open strSQL, strConnStr, adOpenStatic, adLockOptimistic, a

  If objRs.RecordCount > 4 Then
    objRs.Move 4 ' move to the fifth record
    blnCanBkmrk = objRs.Supports(adBookmark)
    If blnCanBkmrk = True Then
      varBookmark = objRs.Bookmark ' record the bookmark
      objRs.MoveLast ' move to a different record
      objRs.Bookmark = varBookmark ' return to the bookmarked record
    End If
  End If
'EndBookmarkEg
```

The [Supports](#) method is covered in more detail later.

Except for the case of cloned **Recordsets**, bookmarks are unique to the **Recordset** in which they were created, even if the same command is used. This means that you cannot use a **Bookmark** obtained from one **Recordset** to move to the same record in a second **Recordset** opened with the same command.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Using Pages

Use the **PageCount** property to determine how many pages of data are in the **Recordset** object. *Pages* are groups of records whose size equals the **PageSize** property setting. Even if the last page is incomplete because there are fewer records than the **PageSize** value, it counts as an additional page in the **PageCount** value. If the **Recordset** object does not support this property, **PageCount** will be -1 to indicate that the **PageCount** is indeterminable.

Use the **PageSize** property to determine how many records make up a logical page of data. Establishing a page size allows you to use the **AbsolutePage** property to move to the first record of a particular page. This is useful in Web-server scenarios when you want to allow the user to page through data, viewing a certain number of records at a time.

This property can be set at any time, and its value will be used for calculating the location of the first record of a particular page.

Use the **AbsolutePage** property to identify the page number on which the current record is located. Again, the provider must support the appropriate functionality for this property to be available.

AbsolutePage is 1-based and equals 1 when the current record is the first record in the **Recordset**. Set this property to move to the first record of a particular page. Obtain the total number of pages from the **PageCount** property.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Recordset Positioning

Use the **AbsolutePosition** property to move to a record, based on its ordinal position in the **Recordset** object, or to determine the ordinal position of the current record. The provider must support the appropriate functionality for this property to be available.

AbsolutePosition is 1-based and equals 1 when the current record is the first record in the **Recordset**. As mentioned previously, you can obtain the total number of records in the **Recordset** object from the **RecordCount** property.

When you set the **AbsolutePosition** property, even if it is to a record in the current cache, ADO reloads the cache with a new group of records starting with the record you specified. The **CacheSize** property determines the size of this group.

Note You should not use the **AbsolutePosition** property as a surrogate record number. The position of a given record changes when you delete a preceding record. There also is no assurance that a given record will have the same **AbsolutePosition** if the **Recordset** object is requeryed or reopened. Bookmarks are the recommended way of retaining and returning to a given position and are the only way of positioning across all types of **Recordset** objects.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Understanding Recordset Structure

Every **Recordset** has a **Fields** collection consisting of one or more **Field** objects. A **Field** object usually represents a table column. The following topics will explain how to navigate through the **Fields** collection and get information about each field. Then they will discuss what kind of information is available to you via the **Field** object and how to use it.

- [The Fields Collection](#)
- [The Field Object](#)
- [Working with Recordsets](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

The Fields Collection

The **Fields** collection is one of ADO's intrinsic collections. A collection is an ordered set of items that can be referred to as a unit. For more information about ADO collections, see [The ADO Object Model](#) in Chapter 1.

The **Fields** collection contains a **Field** object for every field (column) in the **Recordset**. Like all ADO collections, it has **Count** and **Item** properties, as well as **Append** and **Refresh** methods. It also has **CancelUpdate**, **Delete**, **Resync**, and **Update** methods, which are not available to other ADO collections.

Examining the Fields Collection

Consider the **Fields** collection of the sample **Recordset** introduced in this chapter. The sample **Recordset** was derived from the SQL statement

```
SELECT ProductID, ProductName, UnitPrice FROM Products WHERE Categor
```

Thus, you should find that the **Recordset Fields** collection contains three fields.

```
'BeginWalkFields
  Dim objFields As ADODB.Fields

  objRs.Open strSQL, strConnStr, adOpenForwardOnly, adLockReadOnly

  Set objFields = objRs.Fields

  For intLoop = 0 To (objFields.Count - 1)
    Debug.Print objFields.Item(intLoop).Name
  Next
'EndWalkFields
```

This code simply determines the number of **Field** objects in the **Fields** collection using the **Count** property and loops through the collection, returning the value of the **Name** property for each **Field** object. You can use many more **Field** properties to get information about a field. For more information about querying a **Field**, see [The Field Object](#).

Counting Columns

As you might expect, the **Count** property returns the actual number of **Field** objects in the **Fields** collection. Because numbering for members of a collection begins with zero, you should always code loops starting with the zero member and ending with the value of the **Count** property minus 1. If you are using Microsoft Visual Basic and want to loop through the members of a collection without checking the **Count** property, use the **For Each...Next** command.

If the **Count** property is zero, there are no objects in the collection.

Getting to the Field

As with any ADO collection, the **Item** property is the default property of the collection. It returns the individual **Field** object specified by the name or index passed to it. Therefore, the following statements are equivalent for the sample **Recordset**:

```
objField = objRecordset.Fields.Item("ProductID")
objField = objRecordset.Fields("ProductID")
objField = objRecordset.Fields.Item(0)
objField = objRecordset.Fields(0)
```

If these methods are equivalent, which is best? It depends. Using an index to retrieve a **Field** from the collection is faster because it accesses the **Field** directly without having to perform a string lookup. On the other hand, the order of **Fields** within the collection must be known, and if the order changes, the reference to the **Field's** index will have to be changed wherever it occurs. Although slightly slower, using the name of the **Field** is more flexible because it doesn't depend on the order of the **Fields** in the collection.

Using the Refresh Method

Unlike some other ADO collections, using the **Refresh** method on the **Fields** collection has no visible effect. To retrieve changes from the underlying database structure, you must use either the **Requery** method, or if the **Recordset** object does not support bookmarks, the **MoveFirst** method, which will cause the command to be executed against the provider again.

Adding Fields to a Recordset

The **Append** method is used to add fields to a **Recordset**.

You can use the **Append** method to fabricate a **Recordset** programmatically without opening a connection to a data source. A run-time error will occur if the **Append** method is called on the **Fields** collection of an open **Recordset** or on a **Recordset** where the **ActiveConnection** property has been set. You can append fields only to a **Recordset** that is not open and has not yet been connected to a data source. However, to specify values for the newly appended **Fields**, the **Recordset** must first be opened.

Developers often need a place to temporarily store some data, or want some data to act as if it came from a server so it can participate in data binding in a user interface. ADO (in conjunction with the [Microsoft Cursor Service for OLE DB](#)) enables the developer to build an empty **Recordset** object by specifying column information and calling **Open**. In the following example, three new fields are appended to a new **Recordset** object. Then the **Recordset** is opened, two new records are added, and the **Recordset** is persisted to a file. (For more information about **Recordset** persistence, see [Chapter 5: Updating and Persisting Data](#).)

```
'BeginFabricate
Dim objRs As New ADODB.Recordset

With objRs.Fields
    .Append "StudentID", adChar, 11, adFldUpdatable
    .Append "FullName", adVarChar, 50, adFldUpdatable
    .Append "PhoneNmbr", adVarChar, 20, adFldUpdatable
End With

With objRs
    .Open

    .AddNew
    .Fields(0) = "123-45-6789"
    .Fields(1) = "John Doe"
    .Fields(2) = "(425) 555-5555"
    .Update

    .AddNew
    .Fields(0) = "123-45-6780"
```

```
.Fields(1) = "Jane Doe"  
.Fields(2) = "(615) 555-1212"  
.Update  
End With  
  
objRs.Save App.Path & "\FabriTest.adtg", adPersistADTG  
  
objRs.Close  
'EndFabricate
```

The usage of the **Fields Append** method differs between the **Recordset** object and the **Record** object. For more information about the **Record** object, see [Chapter 10: Records and Streams](#).

See Also

[Fabricating Hierarchical Recordsets](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

The Field Object

Each **Field** object usually corresponds to a column in a database table. However, a **Field** can also represent a pointer to another **Recordset**, called a chapter. Exceptions, such as chapter columns, will be covered later in this guide.

Use the **Value** property of **Field** objects to set or return data for the current record. Depending on the functionality the provider exposes, some collections, methods, or properties of a **Field** object may not be available.

With the collections, methods, and properties of a **Field** object, you can do the following:

- Return the name of a field by using the **Name** property.
- View or change the data in the field by using the **Value** property. **Value** is the default property of the **Field** object.
- Return the basic characteristics of a field by using the **Type**, **Precision**, and **NumericScale** properties.
- Return the declared size of a field by using the **DefinedSize** property.
- Return the actual size of the data in a given field by using the **ActualSize** property.
- Determine what types of functionality are supported for a given field by using the **Attributes** property and **Properties** collection.
- Manipulate the values of fields containing long binary or long character data by using the **AppendChunk** and **GetChunk** methods.

Resolve discrepancies in field values during batch updating by using the **OriginalValue** and **UnderlyingValue** properties, if the provider supports batch updates.

Describing a Field

The topics that follow will discuss properties of the [Field](#) object that represent information that describes the **Field** object itself—that is, metadata about the field. This information can be used to determine much about the schema of the **Recordset**. These properties include **Type**, **DefinedSize** and **ActualSize**, **Name**, and **NumericScale** and **Precision**.

Discovering the Data Type

The **Type** property indicates the data type of the field. The data type enumerated constants that are supported by ADO are described in [DataTypeEnum](#) in the *ADO Programmer's Reference*.

For floating point numeric types such **adNumeric**, you can obtain more information. The **NumericScale** property indicates how many digits to the right of the decimal point will be used to represent values for the **Field**. The **Precision** property specifies the maximum number of digits used to represent values for the **Field**.

Determining Field Size

Use the **DefinedSize** property to determine the data capacity of a **Field** object.

Use the **ActualSize** property to return the actual length of a **Field** object's value. For all fields, the **ActualSize** property is read-only. If ADO cannot determine the length of the **Field** object's value, the **ActualSize** property returns **adUnknown**.

The **DefinedSize** and **ActualSize** properties have different purposes. For example, consider a **Field** object with a declared type of **adVarChar** and a **DefinedSize** property value of 50, containing a single character. The **ActualSize** property value it returns is the length in bytes of the single character.

Determining Field Contents

The identifier of the column from the data source is represented by the **Name** property of the **Field**. The **Value** property of the **Field** object returns or sets the

actual data content of the field. This is the default property.

To change the data in a field, set the **Value** property equal to a new value of the correct type. Your cursor type must support updates to change the contents of a field. Database validation is not done here in batch mode, so you will need to check for errors when you call **UpdateBatch** in such a case. Some providers also support the ADO **Field** object's **UnderlyingValue** and **OriginalValue** properties to assist you with resolving conflicts when you attempt to perform batch updates. For details about how to resolve such conflicts, see [Chapter 4: Editing Data](#).

Note **Recordset Field** values cannot be set when appending new **Fields** to a **Recordset**. Rather, new **Fields** can be appended to a closed **Recordset**. Then the **Recordset** must be opened, and only then can values be assigned to these **Fields**.

Getting More Field Information

ADO objects have two types of properties: built-in and dynamic. To this point, only the built-in properties of the **Field** object have been discussed.

Built-in properties are those properties implemented in ADO and immediately available to any new object, using the `MyObject.Property` syntax. They do not appear as **Property** objects in an object's **Properties** collection.

Dynamic properties are defined by the underlying data provider, and appear in the **Properties** collection for the appropriate ADO object. For example, a property specific to the provider may indicate if a **Recordset** object supports transactions or updating. These additional properties will appear as **Property** objects in that **Recordset** object's **Properties** collection. Dynamic properties can be referenced only through the collection, using the syntax `MyObject.Properties(0)` or `MyObject.Properties("Name")`.

You cannot delete either kind of property.

A dynamic **Property** object has four built-in properties of its own:

- The **Name** property is a string that identifies the property.
- The **Type** property is an integer that specifies the property data type.

- The **Value** property is a variant that contains the property setting. **Value** is the default property for a **Property** object.
- The **Attributes** property is a **Long** value that indicates characteristics of the property specific to the provider.

The **Properties** collection for the **Field** object contains additional metadata about the field. The contents of this collection vary depending upon the provider. The following code example examines the **Properties** collection of the sample **Recordset** introduced at the beginning of this chapter. It first looks at the contents of the collection. This code uses the [OLE DB Provider for SQL Server](#), so the **Properties** collection contains information relevant to that provider.

```
'BeginFieldProps
  Dim objProp As ADODB.Property

  For intLoop = 0 To (objFields.Count - 1)
    Debug.Print objFields.Item(intLoop).Name

    For Each objProp In objFields(intLoop).Properties
      Debug.Print vbTab & objProp.Name & " = " & objProp.Value
    Next objProp
  Next intLoop
'EndFieldProps
```

Dealing with Binary Data

Use the [AppendChunk](#) method on a **Field** object to fill it with long binary or character data. In situations where system memory is limited, you can use the **AppendChunk** method to manipulate long values in portions rather than in their entirety.

If the **adFldLong** bit in the **Attributes** property of a **Field** object is set to **True**, you can use the **AppendChunk** method for that field.

The first **AppendChunk** call on a **Field** object writes data to the field, overwriting any existing data. Subsequent **AppendChunk** calls add to existing data. If you are appending data to one field and then you set or read the value of another field in the current record, ADO assumes that you are finished appending data to the first field. If you call the **AppendChunk** method on the first field again, ADO interprets the call as a new **AppendChunk** operation and overwrites the existing data. Accessing fields in other **Recordset** objects that are

not clones of the first **Recordset** object will not disrupt **AppendChunk** operations.

Use the **GetChunk** method on a **Field** object to retrieve part or all of its long binary or character data. In situations where system memory is limited, you can use the **GetChunk** method to manipulate long values in portions, rather than in their entirety.

The data that a **GetChunk** call returns is assigned to *variable*. If *Size* is greater than the remaining data, the **GetChunk** method returns only the remaining data without padding *variable* with empty spaces. If the field is empty, the **GetChunk** method returns a null value.

Each subsequent **GetChunk** call retrieves data starting from where the previous **GetChunk** call left off. However, if you are retrieving data from one field and then set or read the value of another field in the current record, ADO assumes you have finished retrieving data from the first field. If you call the **GetChunk** method on the first field again, ADO interprets the call as a new **GetChunk** operation and starts reading from the beginning of the data. Accessing fields in other **Recordset** objects that are not clones of the first **Recordset** object will not disrupt **GetChunk** operations.

If the **adFldLong** bit in the **Attributes** property of a **Field** object is set to **True**, you can use the **GetChunk** method for that field.

If there is no current record when you use the **GetChunk** or **AppendChunk** method on a **Field** object, error 3021 (no current record) occurs.

For an example of using these methods to manipulate binary data, see the [AppendChunk Method](#) and [GetChunk Method](#) examples in the *ADO Programmer's Reference*.

ADO 2.5 

Working with Recordsets

The **Recordset** object has built-in features that make it possible for you to rearrange the order of the data in the result set, to search for a specific record based on criteria that you supply, and even to optimize those search operations using indexes. Whether these features are available for use depends on the provider and in some cases—such as that of the [Index](#) property—the structure of the data source itself.

Arranging Data

Often, the most efficient way to order the data in your **Recordset** is by specifying an ORDER BY clause in the SQL command used to return results to it. However, you might need to change the order of the data in a **Recordset** that has already been created. You can use the **Sort** property to establish the order in which rows of a **Recordset** are traversed. Furthermore, the **Filter** property determines which rows are accessible when traversing rows.

The **Sort** property sets or returns a **String** value that indicates the field names in the **Recordset** on which to sort. Each name is separated by a comma and is optionally followed by a space and the keyword **ASC** (which sorts the field in ascending order) or **DESC** (which sorts the field in descending order). By default, if no keyword is specified, the field is sorted in ascending order.

The sort operation is efficient because data is not physically rearranged but is simply accessed in the order specified by an index.

The **Sort** property requires the [CursorLocation](#) property to be set to **adUseClient**. A temporary index will be created for each field specified in the **Sort** property if an index does not already exist.

Setting the **Sort** property to an empty string will reset the rows to their original order and delete temporary indexes. Existing indexes will not be deleted.

Suppose a **Recordset** contains three fields named *firstName*, *middleInitial*, and *lastName*. Set the **Sort** property to the string "lastName DESC, firstName ASC", which will order the **Recordset** by last name in descending order and then by first name in ascending order. The middle initial is ignored.

No field referenced in a sort criteria string can be named "ASC" or "DESC" because those names conflict with the keywords **ASC** and **DESC**. Give a field with a conflicting name an alias by using the **AS** keyword in the query that returns the **Recordset**.

For more details about **Recordset** filtering, see Filtering the Results later in this topic.

Finding a Specific Record

ADO provides the [Find](#) and [Seek](#) methods for locating a particular record in a **Recordset**. The **Find** method is supported by a variety of providers but is limited to a single search criterion. The **Seek** method supports searching on multiple criteria, but is not supported by many providers.

Indexes on fields can greatly enhance the performance of the **Recordset** object's **Find** method and **Sort** and **Filter** properties. You can create an internal index for a **Field** object by setting its dynamic [Optimize](#) property. This dynamic property is added to the **Field** object's **Properties** collection when you set the [CursorLocation](#) property to **adUseClient**. Remember that this index is internal to ADO—you cannot gain access to it or use it for any other purpose. Also, this index is distinct from the **Recordset** object's [Index](#) property.

The **Find** method quickly locates a value within a column (field) of a **Recordset**. You can often improve the speed of the **Find** method's operation on a column by using the **Optimize** property to create an index on it.

The **Find** method limits your search to the contents of one field. The **Seek** method requires that you have an index and has other limitations as well. If you need to search on multiple fields that aren't the basis of an index, or if your provider does not support indexes, you can limit your results using the **Filter** property of the **Recordset** object.

Find

The **Find** method searches a **Recordset** for the row that satisfies a specified criterion. Optionally, the direction of the search, starting row, and offset from the starting row may be specified. If the criterion is met, the current row position is set on the found record; otherwise, the position is set to the end (or start) of the **Recordset**, depending on search direction.

Only a single-column name may be specified for the criterion. In other words, this method does not support multi-column searches.

The comparison operator for the criterion may be ">" (greater than), "<" (less than), "=" (equal), ">=" (greater than or equal), "<=" (less than or equal), "<>"

(not equal), or "LIKE" (pattern matching).

The criterion value may be a string, floating-point number, or date. String values are delimited with single quotes or "#" (number sign) marks (for example, "state = 'WA'" or "state = #WA#"). Date values are delimited with "#" (number sign) marks (for example, "start_date > #7/22/97#").

If the comparison operator is "like", the string value may contain an asterisk (*) to find one or more occurrences of any character or substring. For example, "state like 'M*'" matches Maine and Massachusetts. You can also use leading and trailing asterisks to find a substring contained within the values. For example, "state like '*as*'" matches Alaska, Arkansas, and Massachusetts.

Asterisks can be used only at the end of a criteria string or together at both the beginning and end of a criteria string, as shown above. You cannot use the asterisk as a leading wildcard ('*str') or embedded wildcard ('s*r'). This will cause an error.

Seek and Index

Use the **Seek** method in conjunction with the **Index** property if the underlying provider supports indexes on the **Recordset** object. Use the [Supports\(adSeek\)](#) method to determine whether the underlying provider supports **Seek**, and the [Supports\(adIndex\)](#) method to determine whether the provider supports indexes. (For example, the [OLE DB Provider for Microsoft Jet](#) supports **Seek** and **Index**.)

If **Seek** does not find the desired row, no error occurs, and the row is positioned at the end of the **Recordset**. Set the **Index** property to the desired index before executing this method.

This method is supported only with server-side cursors. **Seek** is not supported when the **Recordset** object's [CursorLocation](#) property value is **adUseClient**.

This method can be used only when the **Recordset** object has been opened with a [CommandTypeEnum](#) value of **adCmdTableDirect**.

Filtering the Results

The **Find** method limits your search to the contents of one field. The **Seek** method requires that you have an index and has other limitations as well. If you need to search on multiple fields that are not the basis of an index or if your provider does not support indexes, you can limit your results using the **Filter** property of the **Recordset** object.

Use the **Filter** property to selectively screen out records in a **Recordset** object. The filtered **Recordset** becomes the current cursor, which means that records that do not satisfy the **Filter** criteria are not available in the **Recordset** until the **Filter** is removed. Other properties that return values based on the current cursor are affected, such as **AbsolutePosition**, **AbsolutePage**, **RecordCount**, and **PageCount**. This is because setting the **Filter** property to a specific value will move the current record to the first record that satisfies the new value.

The **Filter** property takes a variant argument. This value represents one of three methods for using the **Filter** property: a criteria string, a **FilterGroupEnum** constant, or an array of bookmarks. For more information, see Filtering with a Criteria String, Filtering with a Constant, and Filtering with Bookmarks later in this topic.

Note When you know the data you want to select, it is usually more efficient to open a **Recordset** with a SQL statement that effectively filters the result set, rather than relying on the **Filter** property.

To remove a filter from a **Recordset**, use the **adFilterNone** constant. Setting the **Filter** property to a zero-length string ("") has the same effect as using the **adFilterNone** constant.

Filtering with a Criteria String

The criteria string is made up of clauses in the form *FieldName Operator Value* (for example, "LastName = 'Smith'"). You can create compound clauses by concatenating individual clauses with AND (for example, "LastName = 'Smith' AND FirstName = 'John'") and OR (for example, "LastName = 'Smith' OR LastName = 'Jones'"). Use the following guidelines for criteria strings:

- *FieldName* must be a valid field name from the **Recordset**. If the field name contains spaces, you must enclose the name in square brackets.
- *Operator* must be one of the following: <, >, <=, >=, <>, =, or LIKE.
- *Value* is the value with which you will compare the field values (for example, 'Smith', #8/24/95#, 12.345, or \$50.00). Use single quotation marks (') with strings and pound signs (#) with dates. For numbers, you can use decimal points, dollar signs, and scientific notation. If *Operator* is LIKE, *Value* can use wildcards. Only the asterisk (*) and percent sign (%) wildcards are allowed, and they must be the last character in the string. *Value* cannot be null.

Note To include single quotation marks (') in the filter *Value*, use two single quotation marks to represent one. For example, to filter on *O'Malley*, the criteria string should be "col1 = 'O'Malley'". To include single quotation marks at both the beginning and the end of the filter value, enclose the string in pound signs (#). For example, to filter on '1', the criteria string should be "col1 = #'1'#".

There is no precedence between AND and OR. Clauses can be grouped within parentheses. However, you cannot group clauses joined by an OR and then join the group to another clause with an AND, like this:

```
(LastName = 'Smith' OR LastName = 'Jones') AND FirstName = 'John'
```

Instead, you would construct this filter as:

```
(LastName = 'Smith' AND FirstName = 'John') OR (LastName = 'Jones' A
```

In a LIKE clause, you can use a wildcard at the beginning and end of the pattern (for example, LastName Like '*mit*') or only at the end of the pattern (for example, LastName Like 'Smit*').

Filtering with a Constant

The following constants are available for filtering **Recordsets**.

Constant	Description
adFilterAffectedRecords	Filters for viewing only records effected by the last Delete , Resync , UpdateBatch , or CancelBatch call.

adFilterConflictingRecords

Filters for viewing the records that failed the last batch update.

adFilterFetchedRecords

Filters for viewing the records in the current cache—that is, the results of the last call to retrieve records from the database.

adFilterNone

Removes the current filter and restores all records for viewing.

adFilterPendingRecords

Filters for viewing only records that have changed but have not yet been sent to the server. Applicable only for batch update mode.

The filter constants make it easier to resolve individual record conflicts during batch update mode by allowing you to view, for example, only those records that were effected during the last **UpdateBatch** method call, as shown in the following example:

```
'BeginDeleteGroup
  'add some bogus records
  With objRs1
    For i = 0 To 8
      .AddNew
      .Fields("CompanyName") = "Shipper Number " & i + 1
      .Fields("Phone") = "(425) 555-000" & (i + 1)
      .Update
    Next i

  're-connect & update
  .ActiveConnection = GetNewConnection
  .UpdateBatch

  'filter on newly added records
  .Filter = adFilterAffectedRecords
  Debug.Print "Deleting the " & .RecordCount & _
    " records you just added."

  'delete the newly added bogus records
  .Delete adAffectGroup
  .Filter = adFilterNone
  Debug.Print .RecordCount & " records remain."

  .Close
End With
```

```
'EndDeleteGroup
```

Filtering with Bookmarks

Finally, you can pass a variant array of bookmarks to the **Filter** property. The resulting cursor will contain only those records whose bookmark was passed in to the property. The following code example creates an array of bookmarks from the records in a **Recordset** which have a "B" in the *ProductName* field. It then passes the array to the **Filter** property and displays information about the resulting filtered **Recordset**.

```
'BeginFilterBkmk
  Dim vBkmkArray() As Variant
  Dim i As Integer

  'Recordset created using "SELECT * FROM Products" as command.
  'So, we will check to see if ProductName has a capital B, and
  'if so, add to the array.
  i = 0
  Do While Not objRs.EOF
    If InStr(1, objRs("ProductName"), "B") Then
      ReDim Preserve vBkmkArray(i)
      vBkmkArray(i) = objRs.Bookmark
      i = i + 1
      Debug.Print objRs("ProductName")
    End If
    objRs.MoveNext
  Loop

  'Filter using the array of bookmarks.
  objRs.Filter = vBkmkArray

  objRs.MoveFirst
  Do While Not objRs.EOF
    Debug.Print objRs("ProductName")
    objRs.MoveNext
  Loop
'EndFilterBkmk
```

Creating a Clone of a Recordset

Use the **Clone** method to create multiple, duplicate **Recordset** objects, particularly if you want to maintain more than one current record in a given set of records. Using the **Clone** method is more efficient than creating and opening a new **Recordset** object with the same definition as the original.

The current record of a newly created clone is originally set to the first record. The current record pointer in a cloned **Recordset** is not synchronized with the original or vice versa. You can navigate independently in each **Recordset**.

Changes you make to one **Recordset** object are visible in all of its clones regardless of [cursor](#) type. However, after you execute [Requery](#) on the original **Recordset**, the clones will no longer be synchronized to the original.

Closing the original **Recordset** does not close its copies, nor does closing a copy close the original or any of the other copies.

You can clone a **Recordset** object only if it supports bookmarks. Bookmark values are interchangeable; that is, a bookmark reference from one **Recordset** object refers to the same record in any of its clones.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Chapter 4: Editing Data

The preceding two chapters explained how use ADO to connect to a data source, execute a command, get the results in a **Recordset** object, and navigate within the **Recordset**. This chapter focuses on the next fundamental ADO operation: editing data.

This chapter continues to use the sample **Recordset** introduced in Chapter 3—with one important change. The following code is used to open the **Recordset**:

```
    . . .  
'BeginEditIntro  
    Dim strSQL As String  
    Dim objRs1 As ADO.DB.Recordset  
  
    strSQL = "SELECT * FROM Shippers"  
  
    Set objRs1 = New ADO.DB.Recordset  
  
    objRs1.Open strSQL, GetNewConnection, adOpenStatic, _  
                adLockBatchOptimistic, adCmdText  
  
    ' Disconnect the Recordset from the Connection object.  
    Set objRs1.ActiveConnection = Nothing  
'EndEditIntro  
  
    . . .
```

The important change to the code involves setting the **Connection** object's **CursorLocation** property equal to **adUseClient** in the `GetNewConnection` function (shown below), which indicates the use of a client cursor. For more information about the differences between client-side and server-side cursors, see [Chapter 8: Understanding Cursors and Locks](#).

The **CursorLocation** property's **adUseClient** setting moves the location of the cursor from the data source (the SQL Server, in this case) to the location of the client code (the desktop workstation). This setting forces ADO to invoke the Client Cursor Engine for OLE DB on the client in order to create and manage the cursor.

You might also have noticed that the **LockType** parameter of the **Open** method changed to **adLockBatchOptimistic**. This opens the cursor in batch mode. (The provider caches multiple changes and writes them to the underlying data source only when you call the **UpdateBatch** method.) Changes made to the **Recordset** will not be updated in the database until the **UpdateBatch** method is called.

Finally, the code in this chapter uses a modified version of the `GetNewConnection` function, introduced in Chapter 2. This version of the function now returns a client-side cursor. The function looks like this:

```
'BeginNewConnection
Public Function GetNewConnection() As ADODB.Connection
    Dim objConn1 As ADODB.Connection
    Set objConn1 = New ADODB.Connection

    strConnStr = "Provider=SQLOLEDB;Initial Catalog=Northwind;" & _
                "Data Source=MySrvr;Integrated Security=SSPI;"

    objConn1.ConnectionString = strConnStr
    objConn1.CursorLocation = adUseClient
    objConn1.Open

    Set GetNewConnection = objConn1
End Function
'EndNewConnection
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Editing Existing Records

To edit existing records, move to the row you want to edit and change the **Value** property of the fields you want to change. For more information about the **Field** object's **Value** property, see [Chapter 3: Examining Data](#). Depending on your cursor type, you will use **Update** or **UpdateBatch** to send changes back to the data source. For more information, see [Chapter 5: Updating and Persisting Data](#).

It is usually more efficient to use a stored procedure with a command object to perform updates, as well as to perform other operations, because a stored procedure does not require the creation of a cursor. For more information about cursors, see [Chapter 8: Understanding Cursors and Locks](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Adding Records

Use the **AddNew** method to create and initialize a new record in an existing **Recordset**. You can use the **Supports** method with a **CursorOptionEnum** value of **adAddNew** to verify whether you can add records to the current **Recordset** object.

After you call the **AddNew** method, the new record becomes the current record and remains current after you call the **Update** method. If the **Recordset** object does not support bookmarks, you might not be able to access the new record once you move to another record. Therefore, depending on your cursor type, you might need to call the **Requery** method to make the new record accessible.

If you call **AddNew** while editing the current record or while adding a new record, ADO calls the **Update** method to save any changes and then creates the new record.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Adding Records Using AddNew

This is the basic syntax of the **AddNew** method:

```
recordset.AddNew FieldList, Values
```

The *FieldList* and *Values* arguments are optional. *FieldList* is either a single name or an array of names or ordinal positions of the fields in the new record.

The *Values* argument is either a single value or an array of values for the fields in the new record.

Typically, when you intend to add a single record, you will call the **AddNew** method without any arguments. Specifically, you will call **AddNew**, set the **Value** of each field in the new record, and then call **Update** and/or **UpdateBatch**. You can ensure that your **Recordset** supports adding new records by using the **Supports** property with the **adAddNew** enumerated constant.

The following code uses this technique to add a new Shipper to the sample **Recordset**. The ShipperID field value is supplied automatically by SQL Server, so the code does not attempt to supply a field value for the new records.

```
'BeginAddNew1.1
  If objRs1.Supports(adAddNew) Then
    With objRs1
      .AddNew
      .Fields("CompanyName") = "Sample Shipper"
      .Fields("Phone") = "(931) 555-6334"
      .Update
    End With
  End If
'EndAddNew1.1
```

Because this code uses a disconnected **Recordset** with a client-side cursor in batch mode, you must reconnect the **Recordset** to the data source with a new **Connection** object before you can call the **UpdateBatch** method to post changes to the database. This is easily done by using the new function **GetNewConnection**.

```
'BeginAddNew1.2
```

```
'Re-establish a Connection and update  
Set objRs1.ActiveConnection = GetNewConnection  
objRs1.UpdateBatch  
'EndAddNew1.2
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Adding Multiple Fields

Occasionally, it might be more efficient to pass in an array of fields and their corresponding values to the **AddNew** method, rather than setting **Value** multiple times for each new field. If *FieldList* is an array, *Values* must also be an array with the same number of members; otherwise, an error occurs. The order of field names must match the order of field values in each array. The following code passes an array of fields and an array of values to the **AddNew** method.

```
'BeginAddNew2
  Dim avarFldNames As Variant
  Dim avarFldValues As Variant

  avarFldNames = Array("CompanyName", "Phone")
  avarFldValues = Array("Sample Shipper 2", "(931) 555-6334")

  If objRs1.Supports(adAddNew) Then
    objRs1.AddNew avarFldNames, avarFldValues
  End If

  'Re-establish a Connection and update
  Set objRs1.ActiveConnection = GetNewConnection
  objRs1.UpdateBatch
'EndAddNew2
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Determining Edit Mode

ADO maintains an editing buffer associated with the current record. The **EditMode** property indicates whether changes have been made to this buffer or whether a new record has been created. Use **EditMode** to determine the editing status of the current record. You can test for pending changes if an editing process has been interrupted and determine whether you need to use the **Update** or **CancelUpdate** method.

EditMode returns one of the **EditModeEnum** constants, which are listed in the following table.

Constant	Description
adEditNone	Indicates that no editing operation is in progress.
adEditInProgress	Indicates that data in the current record has been modified but not saved.
adEditAdd	Indicates that the AddNew method has been called, and the current record in the copy buffer is a new record that has not been saved to the database.
adEditDelete	Indicates that the current record has been deleted.

EditMode can return a valid value only if there is a current record. **EditMode** will return an error if **BOF** or **EOF** is **True** or if the current record has been deleted.

ADO 2.5 

Using AddNew in Immediate and Batch Modes

The behavior of the **AddNew** method depends on the updating mode of the **Recordset** object and whether you pass the *FieldList* and *Values* arguments.

In immediate update mode (in which the provider writes changes to the underlying data source once you call the **Update** method), calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. The provider caches any field value changes locally. Calling the **Update** method posts the new record to the database and resets the **EditMode** property to **adEditNone**. If you pass the *FieldList* and *Values* arguments, ADO immediately posts the new record to the database (no **Update** call is necessary); the **EditMode** property value does not change (**adEditNone**).

In batch update mode, calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. The provider caches any field value changes locally. Calling the **Update** method adds the new record to the current **Recordset** and resets the **EditMode** property to **adEditNone**, but the provider does not post the changes to the underlying database until you call the **UpdateBatch** method. If you pass the *FieldList* and *Values* arguments, ADO sends the new record to the provider for storage in a cache; you need to call the **UpdateBatch** method to post the new record to the underlying database. For more information about **Update** and **UpdateBatch**, see [Chapter 5: Updating and Persisting Data](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Determining What is Supported

The **Supports** method is used to determine whether a specified **Recordset** object supports a particular type of functionality. It has the following syntax:

```
boolean = recordset.Supports( CursorOptions )
```

Supports returns a Boolean value that indicates whether all of the features identified by the *CursorOptions* argument are supported by the provider. You can use the **Supports** method to determine what types of functionality a **Recordset** object supports. If the **Recordset** object supports the features whose corresponding constants are in *CursorOptions*, the **Supports** method returns **True**. Otherwise, it returns **False**.

Using the **Supports** method, you can check for the ability of the **Recordset** object to add new records, use bookmarks, use the **Find** method, use scrolling, use the **Index** property, and to perform batch updates. For a complete list of constants and their meanings, see [CursorOptionEnum](#).

Although the **Supports** method may return **True** for a given functionality, it does not guarantee that the provider can make the feature available under all circumstances. The **Supports** method simply returns whether the provider can support the specified functionality, assuming certain conditions are met. For example, the **Supports** method may indicate that a **Recordset** object supports updates, even though the cursor is based on a multiple table join—some columns of which are not updateable.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Deleting Records Using the Delete Method

Using the **Delete** method marks the current record or a group of records in a **Recordset** object for deletion. If the **Recordset** object does not allow record deletion, an error occurs. If you are in immediate update mode, deletions occur in the database immediately. If a record cannot be successfully deleted (due to database integrity violations, for example), the record will remain in edit mode after the call to **Update**. This means that you must cancel the update using [CancelUpdate](#) before moving off the current record (for example, using [Close](#), [Move](#), or [NextRecordset](#)).

If you are in batch update mode, the records are marked for deletion from the cache and the actual deletion happens when you call the **UpdateBatch** method. (To view the deleted records, set the **Filter** property to **adFilterAffectedRecords** after **Delete** is called.)

Attempting to retrieve field values from the deleted record generates an error. After deleting the current record, the deleted record remains current until you move to a different record. Once you move away from the deleted record, it is no longer accessible.

If you nest deletions in a transaction, you can recover deleted records by using the **RollbackTrans** method. If you are in batch update mode, you can cancel a pending deletion or group of pending deletions by using the **CancelBatch** method.

If the attempt to delete records fails because of a conflict with the underlying data (for example, a record has already been deleted by another user), the provider returns warnings to the **Errors** collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records.

If the **Unique Table** dynamic property is set and the **Recordset** is the result of executing a JOIN operation on multiple tables, the **Delete** method will delete rows only from the table named in the **Unique Table** property.

The **Delete** method takes an optional argument that allows you to specify which records are affected by the **Delete** operation. The only valid values for this argument are either of the following ADO **AffectEnum** enumerated constants:

- **adAffectCurrent** Affects only the current record.
- **adAffectGroup** Affects only records that satisfy the current **Filter** property setting. The **Filter** property must be set to a **FilterGroupEnum** value or an array of **Bookmarks** to use this option.

The following code shows an example of specifying **adAffectGroup** when calling the **Delete** method. This example adds some records to the sample **Recordset** and updates the database. Then it filters the **Recordset** using the **adFilterAffectedRecords** filter enumerated constant, which leaves only the newly added records visible in the **Recordset**. Finally, it calls the **Delete** method and specifies that all of the records that satisfy the current **Filter** property setting (the new records) should be deleted.

```
'BeginDeleteGroup
  'add some bogus records
  With objRs1
    For i = 0 To 8
      .AddNew
      .Fields("CompanyName") = "Shipper Number " & i + 1
      .Fields("Phone") = "(425) 555-000" & (i + 1)
      .Update
    Next i

    're-connect & update
    .ActiveConnection = GetNewConnection
    .UpdateBatch

    'filter on newly added records
    .Filter = adFilterAffectedRecords
    Debug.Print "Deleting the " & .RecordCount & _
      " records you just added."

    'delete the newly added bogus records
    .Delete adAffectGroup
    .Filter = adFilterNone
    Debug.Print .RecordCount & " records remain."

    .Close
  End With
'EndDeleteGroup
```

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

Alternatives: Using SQL Statements

ADO also allows using commands as alternatives to its built-in properties and methods for editing data. Depending upon your provider, all operations mentioned in this chapter could also be accomplished by passing commands to your data source. For example, SQL UPDATE statements can be used to modify data without using the **Value** property of a **Field**. SQL INSERT statements can be used to add new records to a data source, rather than the ADO method **AddNew**. For more information about SQL or the data-manipulation language of your provider, see the documentation of your data source.

For example, you can pass a SQL string containing a DELETE statement to a database, as shown in the following code:

```
'BeginSQLDelete  
strSQL = "DELETE FROM Shippers WHERE ShipperID = " & intId  
objConn.Execute strSQL, , adCmdText + adExecuteNoRecords  
'EndSQLDelete
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Chapter 5: Updating and Persisting Data

The preceding chapters have discussed how to use ADO to get to data in a data source, how to move around in the data, and even how to edit the data. Of course, if the goal of your application is to allow users to make changes to the data, you will need to understand how to save those changes. You can either persist the **Recordset** changes to a file using the **Save** method, or you can send the changes back to the data source for storage using the **Update** or **UpdateBatch** methods.

In the preceding chapters, you changed the data in several rows of the **Recordset**. ADO supports two basic notions relating to the addition, deletion, and modification of rows of data.

The first notion is that changes aren't immediately made to the **Recordset**; instead, they are made to an internal *copy buffer*. If you decide you don't want the changes, the modifications in the copy buffer are discarded. If you decide to keep the changes, the changes in the copy buffer are applied to the **Recordset**.

The second notion is that changes are either propagated to the data source as soon as you declare the work on a row complete (that is, *immediate* mode), or all changes to a set of rows are collected until you declare the work for the set complete (that is, *batch* mode). The **LockType** property determines when the changes are made to the underlying data source. **adLockOptimistic** or **adLockPessimistic** specifies immediate mode, while **adLockBatchOptimistic** specifies batch mode. The **CursorLocation** property can affect which **LockType** settings are available. For instance, the **adLockPessimistic** setting is not supported if the **CursorLocation** property is set to **adUseClient**.

In immediate mode, each invocation of the **Update** method propagates the changes to the data source. In batch mode, each invocation of **Update** or movement of the current row position saves the changes to the copy buffer, but only the **UpdateBatch** method propagates the changes to the data source.

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

Updating Data

Update behavior and functionality is largely dependent upon update mode (lock type), cursor type, and cursor location.

Use the **Update** method to save any changes you have made to the current record of a **Recordset** object since calling the **AddNew** method or since changing any field values in an existing record. The **Recordset** object must support updates.

If the **Recordset** object supports batch updating, you can cache multiple changes to one or more records locally until you call the **UpdateBatch** method. If you are editing the current record or adding a new record when you call the **UpdateBatch** method, ADO will automatically call the **Update** method to save any pending changes to the current record before transmitting the batched changes to the provider.

The current record remains current after you call the **Update** or **UpdateBatch** methods.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Immediate Mode

Immediate mode is in effect when the **LockType** property is set to **adLockOptimistic** or **adLockPessimistic**. In immediate mode, changes to a record are propagated to the data source as soon as you declare the work on a row complete by calling the **Update** method.

Calling Update

If you move from the record you are adding or editing before calling the **Update** method, ADO will automatically call **Update** to save the changes. You must call the **CancelUpdate** method before navigation if you want to cancel any changes made to the current record or discard a newly added record.

The current record remains current after you call the **Update** method.

CancelUpdate

Use the **CancelUpdate** method to cancel any changes made to the current row or to discard a newly added row. You cannot cancel changes to the current row or a new row after you call the **Update** method, unless the changes are either part of a transaction that you can roll back with the **RollbackTrans** method or part of a batch update. In the case of a batch update, you can cancel the **Update** with the **CancelUpdate** or **CancelBatch** method.

If you are adding a new row when you call the **CancelUpdate** method, the current row becomes the row that was current before the **AddNew** call.

If you have not changed the current row or added a new row, calling the **CancelUpdate** method generates an error.

ADO 2.5 

Batch Mode

Batch mode is in effect when the **LockType** property is set to **adLockBatchOptimistic** and batch updating is supported by the provider. Certain lock type settings are not available depending on cursor location. For instance, a pessimistic lock type is not available when the **CursorLocation** is set to **adUseClient**. Conversely, a provider may not support a batch optimistic lock when the cursor location is on the server. You should use batch updating with either a keyset or static cursor only.

The **UpdateBatch** method is used to send **Recordset** changes held in the copy buffer to the server to update the data source. In the following section, we will open a **Recordset** in batch mode, make changes to the copy buffer, and then send our changes to the data source using a call to **UpdateBatch**.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Sending the Updates: UpdateBatch Method

The following code opens a **Recordset** in batch mode by setting the **LockType** property to **adLockBatchOptimistic** and the **CursorLocation** to **adUseClient**. It adds two new records and changes the value of a field in an existing record, saving the original values, and then calls **UpdateBatch** to send the changes back to the data source.

```
'BeginBatchUpdate
  strSQL = "SELECT ShipperId, CompanyName, Phone FROM Shippers"

  objRs1.CursorLocation = adUseClient
  objRs1.Open strSQL, strConn, adOpenStatic, adLockBatchOptimistic

  ' Change value of Phone field for first record in Recordset, sav
  ' for later restoration.
  intId = objRs1("ShipperId")
  strPhone = objRs1("Phone")

  objRs1("Phone") = "(111) 555-1111"

  'Add two new records
  For i = 0 To 1
    objRs1.AddNew
    objRs1(1) = "New Shipper #" & CStr((i + 1))
    objRs1(2) = "(nnn) 555-" & i & i & i & i
  Next i

  ' Send the updates
  objRs1.UpdateBatch
'EndBatchUpdate
```

If you are editing the current record or adding a new record when you call the **UpdateBatch** method, ADO will automatically call the **Update** method to save any pending changes to the current record before transmitting the batched changes to the provider.

ADO 2.5 

Filtering for Updated Records

Before you call **UpdateBatch**, you can use the **Recordset Filter** property to view only those records which have been changed since the **Recordset** was opened or the last call to **UpdateBatch**. To do this, set **Filter** equal to **adFilterPendingRecords** to determine how many records will be updated, as shown below.

This example extends the previous **UpdateBatch** example by filtering the **Recordset** just before calling the **UpdateBatch**, showing the user which records will change and allowing her to cancel the update (using the **CancelBatch** method).

```
'BeginFilterAffected
  objRs1.Filter = adFilterPendingRecords
  objRs1.MoveFirst

  strMsg = "The following " & objRs1.RecordCount & " values will "
           "be updated. Do you wish to proceed?"
  While Not objRs1.EOF
    strMsg = strMsg & vbCrLf & objRs1(0) & vbTab & objRs1(1) & v
              objRs1(2) & vbCrLf
    objRs1.MoveNext
  Wend

  blnResp = MsgBox(strMsg, vbYesNo, "Proceed with Update?")
  If blnResp = True Then
    objRs1.UpdateBatch
  Else
    objRs1.CancelBatch
  End If
'EndFilterAffected
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Dealing with Failed Updates

When an update concludes with errors, how you resolve the errors depends on the nature and severity of the errors and the logic of your application. However, if the database is shared with other users, a typical error is that someone else modifies the field before you do. This type of error is called a *conflict*. ADO detects this situation and reports an error.

If there are update errors, they will be trapped in an error-handling routine. Filter the **Recordset** with the **adFilterConflictingRecords** constant so that only the conflicting rows are visible. In this example, the error-resolution strategy is merely to print the author's first and last names (**au_fname** and **au_lname**).

The code to alert the user to the update conflict looks like this:

```
objRs.Filter = adFilterConflictingRecords
objRs.MoveFirst
Do While Not objRst.EOF
    Debug.Print "Conflict: Name = "; objRs!au_fname; " "; objRs!au_l
    objRs.MoveNext
Loop
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Detecting and Resolving Conflicts

If you are dealing with your **Recordset** in immediate mode, there is much less chance for concurrency problems to arise. On the other hand, if your application uses batch mode updating, there may be a good chance that one user will change a record before changes made by another user editing the same record are saved. In such a case, you will want your application to gracefully handle the conflict. It may be your wish that the last person to send an update to the server "wins." Or you may want to let the most recent user to decide which update should take precedence by providing him with a choice between the two conflicting values.

Whatever the case, ADO provides the **UnderlyingValue** and **OriginalValue** properties of the **Field** object in order to handle these types of conflicts. Use these properties in combination with the **Resync** method and **Filter** property of the **Recordset**.

Detecting Errors

When ADO encounters a conflict during a batch update, a warning will be placed in the **Errors** collection. Therefore, you should always check for errors immediately after calling **BatchUpdate**, and if you find them, begin testing the assumption that you have encountered a conflict. The first step is to set the **Filter** property on the **Recordset** equal to **adFilterConflictingRecords** (the **Filter** property is discussed in the preceding chapter). This limits the view on your **Recordset** to only those records that are in conflict. If the **RecordCount** property is equal to zero after this step, you know the error was raised by something other than a conflict.

When you call **BatchUpdate**, ADO and the provider are generating SQL statements to perform updates on the data source. Remember that certain data sources have limitations on which types of columns can be used in a WHERE clause.

Next, call the **Resync** method on the **Recordset** with the *AffectRecords* argument set equal to **adAffectGroup** and the *ResyncValues* argument set equal to **adResyncUnderlyingValues**. The **Resync** method refreshes the data in the current **Recordset** object from the underlying database. By using

adAffectGroup, you are ensuring that only the records visible with the current filter setting, that is, only the conflicting records, are resynchronized with the database. This could make a significant performance difference if you are dealing with a large **Recordset**. By setting the *ResyncValues* argument to **adResyncUnderlyingValues** when calling **Resync**, you ensure that the **UnderlyingValue** property will contain the (conflicting) value from the database, that the **Value** property will maintain the value entered by the user, and that the **OriginalValue** property will hold the original value for the field (the value it had before the last successful **UpdateBatch** call was made). You can then use these values to resolve the conflict programmatically or require the user to choose the value that will be used.

This technique is shown in the code example below. The example artificially creates a conflict by using a separate **Recordset** to change a value in the underlying table before **UpdateBatch** is called.

```
'BeginConflicts
  On Error GoTo ErrHandler:

  Dim objRs1 As New ADODB.Recordset
  Dim objRs2 As New ADODB.Recordset
  Dim strSQL As String
  Dim strMsg As String

  strSQL = "SELECT * FROM Shippers WHERE ShipperID = 2"

  'Open Rs and change a value
  objRs1.CursorLocation = adUseClient
  objRs1.Open strSQL, strConn, adOpenStatic, adLockBatchOptimistic
  objRs1("Phone") = "(111) 555-1111"

  'Introduce a conflict at the db...
  objRs2.Open strSQL, strConn, adOpenKeyset, adLockOptimistic, adC
  objRs2("Phone") = "(999) 555-9999"
  objRs2.Update
  objRs2.Close
  Set objRs2 = Nothing

  On Error Resume Next
  objRs1.UpdateBatch

  If objRs1.ActiveConnection.Errors.Count <> 0 Then
    Dim intConflicts As Integer

    intConflicts = 0
```

```

objRs1.Filter = adFilterConflictingRecords

intConflicts = objRs1.RecordCount

'Resync so we can see the UnderlyingValue and offer user a c
'This sample only displays all three values and resets to or
objRs1.Resync adAffectGroup, adResyncUnderlyingValues

If intConflicts > 0 Then
    strMsg = "A conflict occurred with updates for " & intCo
        " record(s)." & vbCrLf & "The values will be re
        " to their original values." & vbCrLf & vbCrLf

    objRs1.MoveFirst
    While Not objRs1.EOF
        strMsg = strMsg & "SHIPPER = " & objRs1("CompanyName
        strMsg = strMsg & "Value = " & objRs1("Phone").Value
        strMsg = strMsg & "UnderlyingValue = " & _
            objRs1("Phone").UnderlyingValue &
        strMsg = strMsg & "OriginalValue = " & _
            objRs1("Phone").OriginalValue & v
        strMsg = strMsg & vbCrLf & "Original value has been

        MsgBox strMsg, vbOKOnly, _
            "Conflict " & objRs1.AbsolutePosition & _
            " of " & intConflicts

        objRs1("Phone").Value = objRs1("Phone").OriginalValu
        objRs1.MoveNext
    Wend

    objRs1.UpdateBatch adAffectGroup
Else
    'Other error occurred. Minimal handling in this example.
    strMsg = "Errors occurred during the update. " & _
        objRs1.ActiveConnection.Errors(0).Number & "
        objRs1.ActiveConnection.Errors(0).Descriptio

End If

On Error GoTo 0
End If

objRs1.MoveFirst

'Clean up
objRs1.Close
Set objRs1 = Nothing
Exit Sub

```

ErrorHandler:

```
If Not objRs1 Is Nothing Then
    If objRs1.State = adStateOpen Then objRs1.Close
    Set objRs1 = Nothing
End If

If Not objRs2 Is Nothing Then
    If objRs2.State = adStateOpen Then objRs2.Close
    Set objRs2 = Nothing
End If

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

'EndConflicts
```

You can use the **Status** property of the current **Record** or of a specific **Field** to determine what kind of a conflict has occurred.

For more detailed information on error handling, see [Chapter 6: Error Handling](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Disconnecting and Reconnecting the Recordset

One of the most powerful features found in ADO is the capability to open a client-side **Recordset** from a data source and then *disconnect* the **Recordset** from the data source. Once the **Recordset** has been disconnected, the connection to the data source can be closed, thereby releasing the resources on the server used to maintain it. You can continue to view and edit the data in the **Recordset** while it is disconnected and later reconnect to the data source and send your updates in batch mode.

To disconnect a **Recordset**, open it with a cursor location of **adUseClient**, and then set the **ActiveConnection** property equal to *Nothing*. (C++ users should set the **ActiveConnection** equal to NULL to disconnect.)

We will use a disconnected **Recordset** later in this chapter when we discuss **Recordset** persistence to address a scenario in which we need to have the data in a **Recordset** available to an application while the client computer is not connected to a network.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Updating JOINed Results: Unique Table

ADO enables you to closely control modifications to a particular base table in a **Recordset** that was formed by a JOIN operation on multiple base tables using the **Unique Table** dynamic property. For details on using **Unique Table**, refer to the ADO Programmer's Reference topics on the **Unique Table** and **Update Resynch** dynamic properties.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Transaction Processing

ADO provides the following methods for controlling transactions: **BeginTrans**, **CommitTrans**, and **RollbackTrans**. Use these methods with a **Connection** object when you want to save or cancel a series of changes made to the source data as a single unit. For example, to transfer money between accounts, you subtract an amount from one and add the same amount to the other. If either update fails, the accounts no longer balance. Making these changes within an open transaction ensures that either all or none of the changes go through.

Note Not all providers support transactions. Verify that the provider-defined property "**Transaction DDL**" appears in the **Connection** object's [Properties](#) collection, indicating that the provider supports transactions. If the provider does not support transactions, calling one of these methods will return an error.

After you call the **BeginTrans** method, the provider will no longer instantaneously commit changes you make until you call **CommitTrans** or **RollbackTrans** to end the transaction.

Calling the **CommitTrans** method saves changes made within an open transaction on the connection and ends the transaction. Calling the **RollbackTrans** method reverses any changes made within an open transaction and ends the transaction. Calling either method when there is no open transaction generates an error.

Depending on the **Connection** object's [Attributes](#) property, calling either the **CommitTrans** or **RollbackTrans** method may automatically start a new transaction. If the **Attributes** property is set to **adXactCommitRetaining**, the provider automatically starts a new transaction after a **CommitTrans** call. If the **Attributes** property is set to **adXactAbortRetaining**, the provider automatically starts a new transaction after a **RollbackTrans** call.

Transaction Isolation Level

Use the **IsolationLevel** property to set the isolation level of a transaction on a **Connection** object. The setting does not take effect until the next time you call

the [BeginTrans](#) method. If the level of isolation you request is unavailable, the [provider](#) may return the next greater level of isolation. Refer to the **IsolationLevel** property in the ADO Programmer's Reference for more details on valid values.

Nested Transactions

For providers that support nested transactions, calling the **BeginTrans** method within an open transaction starts a new, nested transaction. The return value indicates the level of nesting: a return value of "1" indicates you have opened a top-level transaction (that is, the transaction is not nested within another transaction), "2" indicates that you have opened a second-level transaction (a transaction nested within a top-level transaction), and so forth. Calling **CommitTrans** or **RollbackTrans** affects only the most recently opened transaction; you must close or roll back the current transaction before you can resolve any higher-level transactions.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Persisting Data

Portable computing (for example, using laptops) has generated the need for applications that can run in both a connected and disconnected state. ADO has added support for this by giving the developer the ability to save a client cursor **Recordset** to disk and reload it later.

There are several scenarios in which you could use this type of feature, including the following:

- **Traveling:** When taking the application on the road, it is vital to supply the ability to make changes and add new records that can then be reconnected to the database later and committed.
- **Infrequently updated lookups:** Often in an application, tables are used as lookups—for example, state tax tables. They are infrequently updated and are read-only. Instead of rereading this data from the server each time the application is started, the application can simply load the data from a locally persisted **Recordset**.

In ADO, to save and load **Recordsets**, use the **Recordset.Save** and **Recordset.Open(,,,adCmdFile)** methods on the ADO **Recordset** object.

You can use the **Recordset Save** method to persist your ADO **Recordset** to a file on a disk. (You can also save a **Recordset** to an ADO **Stream** object. **Stream** objects are discussed later in the guide.) Later, you can use the **Open** method to reopen the **Recordset** when you are ready to use it. By default, ADO saves the **Recordset** into the proprietary Microsoft Advanced Data TableGram (ADTG) format. This binary format is specified using the **adPersistADTG PersistFormatEnum** value. Alternatively, you may choose to save your **Recordset** out as XML instead using **adPersistXML**. For more information about saving Recordsets as XML, see [Persisting Records in XML Format](#).

The syntax of the **Save** method is as follows:

```
recordset.Save Destination, PersistFormat
```

The first time you save the **Recordset**, it is optional to specify *Destination*. If

you omit *Destination*, a new file will be created with a name set to the value of the [Source](#) property of the **Recordset**.

Omit *Destination* when you subsequently call **Save** after the first save or a run-time error will occur. If you subsequently call **Save** with a new *Destination*, the **Recordset** is saved to the new destination. However, the new destination and the original destination will both be open.

Save does not close the **Recordset** or *Destination*, so you can continue to work with the **Recordset** and save your most recent changes. *Destination* remains open until the **Recordset** is closed, during which time other applications can read but not write to *Destination*.

For reasons of security, the **Save** method permits only the use of low and custom security settings from a script executed by Microsoft Internet Explorer. For a more detailed explanation of security issues, see "ADO and RDS Security Issues in Microsoft Internet Explorer" under ActiveX Data Objects (ADO) Technical Articles in Microsoft Data Access Technical Articles.

If the **Save** method is called while an asynchronous **Recordset** fetch, execute, or update operation is in progress, **Save** waits until the [asynchronous](#) operation is complete.

Records are saved beginning with the first row of the **Recordset**. When the **Save** method is finished, the current row position is moved to the first row of the **Recordset**.

For best results, set the [CursorLocation](#) property to **adUseClient** with **Save**. If your [provider](#) does not support all of the functionality necessary to save **Recordset** objects, the Cursor Service will provide that functionality.

When a **Recordset** is persisted with the **CursorLocation** property set to **adUseServer**, the update capability for the **Recordset** is limited. Typically, only single-table updates, insertions, and deletions are allowed (dependent on provider functionality). The [Resync](#) method is also unavailable in this configuration.

Because the *Destination* parameter can accept any object that supports the OLE DB **IStream** interface, you can save a **Recordset** directly to the ASP **Response** object.

In the following example, the **Save** and **Open** methods are used to persist a **Recordset** and later reopen it:

```
'BeginPersist
  conn.ConnectionString = _
  "Provider='SQLOLEDB';Data Source='MySqlServer';" _
    & "Integrated Security='SSPI';Initial Catalog='pubs'"
  conn.Open

  conn.Execute "create table testtable (dbkey int " & _
    "primary key, field1 char(10))"
  conn.Execute "insert into testtable values (1, 'string1')"
```

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

More About Recordset Persistence

The ADO Recordset object supports storing a **Recordset** object's contents in a file using its [Save](#) method. The persistently stored file may exist on a local drive, network server, or as a URL on a Web site. Later, the file can be restored with either the **Recordset** object's [Open](#) method or the [Connection](#) object's [Execute](#) method.

In addition, the [GetString](#) method converts a **Recordset** object to a form in which the columns and rows are delimited with characters you specify.

To persist a **Recordset**, begin by converting it to a form that can be stored in a file. **Recordset** objects can be stored in the proprietary Advanced Data TableGram (ADTG) format or the open Extensible Markup Language (XML) format. ADTG examples are shown below. For more information about XML persistence, see [Persisting Records in XML format](#).

Save any pending changes in the persisted file. Doing this allows you to issue a query that returns a **Recordset** object, edits the **Recordset**, saves it and the pending changes, later restores the **Recordset**, and then updates the data source with the saved pending changes.

For information about persistently storing **Stream** objects, see [Streams and Persistence](#) in Chapter 10.

For an example of **Recordset** persistence, see the [XML Recordset Persistence Scenario](#).

Example

Save a Recordset:

```
Dim rs as New ADODB.Recordset
rs.Save "c:\yourFile.adtg", adPersistADTG
```

Open a persisted file with Recordset.Open:

```
Dim rs as New ADODB.Recordset
```

```
rs.Open "c:\yourFile.adtg", "Provider='MSPersist'",,,adCmdFile
```

Optionally, if the **Recordset** does not have an active connection, you can accept all the defaults and simply code the following:

```
Dim rs as New ADODB.Recordset  
rs.Open "c:\yourFile.adtg"
```

Open a persisted file with Connection.Execute:

```
Dim conn as New ADODB.Connection  
Dim rs as ADODB.Recordset  
conn.Open "Provider='MSPersist'"  
Set rs = conn.execute("c:\yourFile.adtg")
```

Open a persisted file with RDS.DataControl:

In this case, the **Server** property is not set.

```
Dim dc as New RDS.DataControl  
dc.Connection = "Provider='MSPersist'"  
dc.SQL = "c:\yourFile.adtg"  
dc.Refresh
```

See Also

[GetString Method](#) | [Microsoft OLE DB Persistence Provider](#) | [Recordset Object](#) | [Streams and Persistence](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Persisting Filtered and Hierarchical Recordsets

If the [Filter](#) property is in effect for the **Recordset**, only the rows accessible under the filter are saved. If the **Recordset** is hierarchical, the current [child Recordset](#) and its children are saved, including the [parent Recordset](#). If the **Save** method of a child **Recordset** is called, the child and all its children are saved, but the parent is not. For more information about hierarchical **Recordsets**, see [Chapter 9: Data Shaping](#).

Note Some limitations apply when saving hierarchical **Recordsets** (data shapes) in XML format. For more information, see [Hierarchical Recordsets in XML](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Persisting Records in XML Format

Like ADTG format, **Recordset** persistence in XML format is implemented with the Microsoft OLE DB Persistence Provider. This provider generates a forward-only, read-only rowset from a saved XML file or stream that contains the schema information generated by ADO. Similarly, it can take an ADO **Recordset**, generate XML, and save it to a file or any object that implements the COM **IStream** interface. (In fact, a file is just another example of an object that supports **IStream**.) For versions 2.5 and later, ADO relies on the Microsoft XML Parser (MSXML) to load the XML into the **Recordset**; therefore msxml.dll is required. For version 2.5, MSXML shipped with Internet Explorer 5. For version 2.6, MSXML shipped with SQL Server 2000.

Note Some limitations apply when saving hierarchical **Recordsets** (data shapes) to XML format. You cannot save to XML if the hierarchical **Recordset** contains pending updates, and you cannot save a parameterized hierarchical **Recordset**. For more information, see [Hierarchical Recordsets in XML](#).

The easiest way to persist data into XML and load it back again through ADO is with the **Save** and **Open** methods, respectively. The following ADO code example demonstrates saving the data in the Titles table to a file named titles.sav.

```
Dim rs as new Recordset
Dim rs2 as new Recordset
Dim c as new Connection
Dim s as new Stream

' Query the Titles table.
c.Open "provider='sqloledb';data source='mydb';initial catalog='pubs
rs.cursorlocation = adUseClient
rs.Open "select * from titles", c, adOpenStatic

' Save to the file in the XML format. Note that if you don't specify
' adPersistXML, a binary format (ADTG) will be used by default.
rs.Save "titles.sav", adPersistXML

' Save the Recordset into the ADO Stream object.
rs.save s, adPersistXML
```

```
rs.Close
c.Close

set rs = nothing

' Reopen the file.
rs.Open "titles.sav",,,,adCmdFile
' Open the Stream back into a Recordset.
rs2.open s
```

ADO always persists the entire **Recordset** object. If you wish to only persist a subset of rows of the **Recordset** object, use the **Filter** method to narrow down the rows or change your selection clause. However, you must open a **Recordset** object with a client-side cursor (**CursorLocation** = **adUseClient**) to use the **Filter** method for saving a subset of rows. For example, to retrieve titles that start with the letter "b," you can apply a filter to an open **Recordset** object:

```
rs.Filter "title_id like 'B*'"
rs.Save "btitles.sav", adPersistXML
```

ADO always uses the Client Cursor Engine rowset to produce a scrollable, bookmarkable **Recordset** object on top of the forward-only data generated by the Persistence Provider.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

XML Persistence Format

ADO uses UTF-8 encoding for the XML stream it persists.

The ADO XML format is broken into two sections, a schema section followed by the data section. The following is an example XML file for the Shippers table from the Northwind database. Various parts of the XML are discussed following the example.

```
<xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
xmlns:rs="urn:schemas-microsoft-com:rowset"
xmlns:z="#RowsetSchema">
  <s:Schema id="RowsetSchema">
    <s:ElementType name="row" content="eltOnly" rs:updatable="true">
      <s:AttributeType name="ShipperID" rs:number="1"
rs:basetable="shippers" rs:basecolumn="ShipperID"
rs:keycolumn="true">
        <s:datatype dt:type="int" dt:maxLength="4" rs:precision="10"
rs:fixedlength="true" rs:maybenull="false"/>
      </s:AttributeType>
      <s:AttributeType name="CompanyName" rs:number="2"
rs:nullable="true" rs:write="true" rs:basetable="shippers"
rs:basecolumn="CompanyName">
        <s:datatype dt:type="string" dt:maxLength="40" />
      </s:AttributeType>
      <s:AttributeType name="Phone" rs:number="3" rs:nullable="true"
rs:write="true" rs:basetable="shippers"
rs:basecolumn="Phone">
        <s:datatype dt:type="string" dt:maxLength="24"/>
      </s:AttributeType>
      <s:extends type="rs:rowbase"/>
    </s:ElementType>
  </s:Schema>

  <rs:data>
    <z:row ShipperID="1" CompanyName="Speedy Express"
Phone="(503) 555-9831"/>
    <z:row ShipperID="2" CompanyName="United Package"
Phone="(503) 555-3199"/>
    <z:row ShipperID="3" CompanyName="Federal Shipping"
Phone="(503) 555-9931"/>
  </rs:data>
</xml>
```

The schema shows the declarations of namespaces, the schema section, and the data section. The schema section contains definitions for row, ShipperID, CompanyName, and Phone.

Schema definitions conform to the XML-Data specification and are able to be fully validated (though validation will not occur in Internet Explorer 5). You can view this specification at <http://www.w3.org/TR/1998/NOTE-XML-data/>. XML-Data is the only supported schema format for **Recordset** persistence currently.

The data section has three rows containing information about shippers. For an empty rowset, the data section may be empty, but the <rs:data> tags must be present. With no data, you could write the tag shorthand as simply <rs:data/>. Any tag prefixed with "rs" indicates that it is in the namespace defined by urn:schemas-microsoft-com:rowset. The full definition of this schema is defined in the appendix to this document.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Namespaces

The XML persistence format in ADO uses the following four namespaces.

Prefix	Description
s	Refers to the "XML-Data" namespace containing the elements and attributes that define the schema of the current Recordset .
dt	Refers to the data type definitions specification.
rs	Refers to the namespace containing elements and attributes specific to ADO Recordset properties and attributes.
z	Refers to the schema of the current rowset.

A client should not add its own tags to these namespaces, as defined by the specification. For example, a client should not define a namespace as "urn:schemas-microsoft-com:rowset" and then write out something such as "rs:MyOwnTag." To learn more about namespaces, see <http://www.w3.org/TR/REC-xml-names/>.

Important The ID for the schema tag must be "RowsetSchema," and the namespace used to refer to the schema of the current rowset must point to "#RowsetSchema."

Note that the prefix of the namespace, that part to the right of the colon and to the left of the equal sign, is arbitrary.

```
xmlns:rs="urn:schemas-microsoft-com:rowset"
```

The user can define this to be any name as long as this name is consistently used throughout the XML document. ADO always writes out "s," "rs," "dt," and "z," but these prefix names are not hard-coded into the loading component.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Schema Section

The schema section is required. As the previous example shows, ADO writes out detailed metadata about each column to preserve the semantics of the data values as much as possible for updating. However, to load in the XML, ADO only requires the names of the columns and the rowset to which they belong. Here is an example of a minimal schema:

```
<xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
      xmlns:rs="urn:schemas-microsoft-com:rowset"
      xmlns:z="#RowsetSchema">
  <s:Schema id="RowsetSchema">
    <s:ElementType name="row" content="eltOnly">
      <s:AttributeType name="ShipperID"/>
      <s:AttributeType name="CompanyName"/>
      <s:AttributeType name="Phone"/>
      <s:Extends type="rs:rowbase"/>
    </s:ElementType>
  </s:Schema>
  <rs:data>
    ...
  </rs:data>
</xml>
```

In the case above, ADO will treat the data as variable length strings because no type information is included in the schema.

Creating Aliases for Column Names

The **rs:name** attribute allows you to create an alias for a column name so that a friendly name may appear in the column information exposed by the rowset and a shorter name may be used in the data section. For example, the schema above could be modified to map ShipperID to s1, CompanyName to s2, and Phone to s3 as follows:

```
<s:Schema id="RowsetSchema">
  <s:ElementType name="row" content="eltOnly" rs:updatable="true">
    <s:AttributeType name="s1" rs:name="ShipperID" rs:number="1" ...>
    ...
  </s:AttributeType>
  <s:AttributeType name="s2" rs:name="CompanyName" rs:number="2" ...>
  ...
</s:Schema>
```

```

</s:AttributeType>
<s:AttributeType name="s3" rs:name="Phone" rs:number="3" ...>
...
</s:AttributeType>
...
</s:ElementType>
</s:Schema>

```

Then, in the data section, the row would use the **name** attribute (not **rs:name**) to refer to that column:

```
"<row s1="1" s2="Speedy Express" s3="(503) 555-9831"/>
```

Creating aliases for column names is required whenever a column name is not a legal attribute or tag name in XML. For example, "LastName" must have an alias because names with embedded spaces are illegal attributes. The following line won't be correctly handled by the XML parser, so you must create an alias to some other name that does not have an embedded space:

```
<row last name="Jones"/>
```

Whatever value you use for the **name** attribute must be used consistently in each place that the column is referenced in both the schema and data sections of the XML document. The following example shows the consistent use of s1:

```

<s:Schema id="RowsetSchema">
  <s:ElementType name="row" content="eltOnly">
    <s:attribute type="s1"/>
    <s:attribute type="CompanyName"/>
    <s:attribute type="s3"/>
    <s:extends type="rs:rowbase"/>
  </s:ElementType>
  <s:AttributeType name="s1" rs:name="ShipperID" rs:number="1"
    rs:maydefer="true" rs:writeunknown="true">
    <s:datatype dt:type="i4" dt:maxLength="4" rs:precision="10"
      rs:fixedlength="true" rs:maybenull="true"/>
  </s:AttributeType>
</s:Schema>
<rs:data>
  <z:row s1="1" CompanyName="Speedy Express" s3="(503) 555-9831"/>
</rs:data>

```

Similarly, because there is no alias defined for CompanyName above, CompanyName must be used consistently throughout the document.

Data Types

You can apply a data type to a column with the **dt:type** attribute. For the definitive guide to allowable XML types, see <http://www.w3.org/TR/1998/NOTE-XML-data-0105/#Datatypes>. You can specify a data type in two ways: either specify the **dt:type** attribute directly on the column definition itself or use the **s:datatype** construct as a nested element of the column definition. For example,

```
<s:AttributeType name="Phone" >
  <s:datatype dt:type="string"/>
</s:AttributeType>
```

is equivalent to

```
<s:AttributeType name="Phone" dt:type="string"/>
```

If you omit the **dt:type** attribute entirely from the row definition, by default, the column's type will be a variable length string.

If you have more type information than simply the type name (for example, **dt:maxLength**), it makes it more readable to use the **s:datatype** child element. This is merely a convention, however, and not a requirement.

The following examples show further how to include type information in your schema:

```
<!-- 1. String with no max length -->
<s:AttributeType name="title_id"/>
<!--or -->
<s:AttributeType name="title_id" dt:type="string"/>

<!-- 2. Fixed length string with max length of 6 -->
<s:AttributeType name="title_id">
  <s:datatype dt:type="string" dt:maxLength="6" rs:fixedlength="tr
</s:AttributeType>

<!-- 3. Variable length string with max length of 6 -->
<s:AttributeType name="title_id">
  <s:datatype dt:type="string" dt:maxLength="6" />
</s:AttributeType>

<!-- 4. Integer -->
<s:AttributeType name="title_id" dt:type="int"/>
```

There is a subtle use of the **rs:fixedlength** attribute in the second example. A column with the **rs:fixedlength** attribute set to true means that the data must have the length defined in the schema. In this case, a legal value for title_id is "123456," as is "123 ." However, "123" would not be valid because its length is 3, not 6. See the OLE DB Programmer's Guide for a more complete description of the **fixedlength** property.

Handling Nulls

Null values are handled by the **rs:maybenull** attribute. If this attribute is set to true, the contents of the column may contain a null value. Furthermore, if the column is not found in a row of data, the user reading the data back from the rowset will get a null status from **IRowset::GetData()**. Consider the following column definitions from the Shippers table:

```
<s:AttributeType name="ShipperID">
  <s:datatype dt:type="int" dt:maxLength="4"/>
</s:AttributeType>
<s:AttributeType name="CompanyName">
  <s:datatype dt:type="string" dt:maxLength="40" rs:maybenull="true"
</s:AttributeType>
```

The definition allows CompanyName to be null, but ShipperID cannot contain a null value. If the data section contained the following row, the Persistence Provider would set the status of the data for the CompanyName column to the OLE DB status constant DBSTATUS_S_ISNULL:

```
<z:row ShipperID="1"/>
```

If the row was entirely empty, as follows, the Persistence Provider would return an OLE DB status of DBSTATUS_E_UNAVAILABLE for ShipperID and DBSTATUS_S_ISNULL for CompanyName.

```
<z:row/>
```

Note that a zero-length string is not the same as null.

```
<z:row ShipperID="1" CompanyName=""/>
```

For the preceding row, the Persistence Provider will return an OLE DB status of DBSTATUS_S_OK for both columns. The CompanyName in this case is simply "" (a zero-length string).

For further information about the OLE DB constructs available for use within the schema of an XML document for OLE DB, see the definition of "urn:schemas-microsoft-com:rowset" and the OLE DB Programmer's Guide.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Data Section

The data section defines the data of the rowset along with any pending updates, insertions, or deletions. The data section may contain zero or more rows. It may only contain data from one rowset where the row is defined by the schema. Also, as noted before, columns without any data may be omitted. If an attribute or subelement is used in the data section and that construct has not been defined in the schema section, it is silently ignored.

String

Reserved XML characters in text data must be replaced with appropriate character entities. For example, in the company name "Joe's Garage," the single quote character must be replaced by an entity. The actual row would look like:

```
<z:row CompanyName="Joe&apos;s Garage"/>
```

The following characters are reserved in XML and must be replaced by character entities: {',",&,<,>}.

Binary

Binary data is bin.hex encoded (that is, one byte maps to two characters, one character per nibble).

DateTime

The variant VT_DATE format is not directly supported by XML-Data data types. The correct format for dates with both a data and time component is yyyy-mm-ddThh:mm:ss.

For more information about date formats specified by XML, refer to <http://www.w3.org/TR/1998/NOTE-XML-data-0105/#Datatypes>.

When the XML-Data specification defines two equivalent data types (for example, i4 == int), ADO will write out the friendly name but read in both.

Managing Pending Changes

A **Recordset** can be opened in immediate or batch update mode. When opened in batch update mode with client-side cursors, all changes made to the **Recordset** are in a pending state until the **UpdateBatch** method is called. Pending changes are also persisted when the **Recordset** is saved. In XML, they are represented by the use of the "update" elements defined in urn:schemas-microsoft-com:rowset. In addition, if a rowset can be updated, the updatable property must be set to true in the definition of the row. For example, to define that the Shippers table contains pending changes, the row definition would look like the following:

```
<s:ElementType name="row" content="eltOnly" updatable="true">
  <s:attribute type="ShipperID"/>
  <s:attribute type="CompanyName"/>
  <s:attribute type="Phone"/>
  <s:extends type="rs:rowbase"/>
</s:ElementType>
```

This tells the Persistence Provider to surface data so that ADO can construct an updatable **Recordset** object.

The following sample data shows how insertions, changes, and deletions look in the persisted file:

```
<rs:data>
  <z:row ShipperID="2" CompanyName="United Package"
    Phone="(503) 555-3199"/>
<rs:update>
  <rs:original>
    <z:row ShipperID="3" CompanyName="Federal Shipping"
      Phone="(503) 555-9931"/>
  </rs:original>
  <z:row Phone="(503) 552-7134"/>
</rs:update>
<rs:insert>
  <z:row ShipperID="12" CompanyName="Lightning Shipping"
    Phone="(505) 111-2222"/>
  <z:row ShipperID="13" CompanyName="Thunder Overnight"
    Phone="(505) 111-2222"/>
  <z:row ShipperID="14" CompanyName="Blue Angel Air Delivery"
    Phone="(505) 111-2222"/>
</rs:insert>
<rs:delete>
  <z:row ShipperID="1" CompanyName="Speedy Express" Phone="(503) 555
```

```
</rs:delete>  
</rs:data>
```

An update always contains the entire original row data followed by the changed row data. The changed row may contain all of the columns or only those columns that have actually changed. In the previous example, the row for Shipper 2 is not changed, while only the Phone column has changed values for Shipper 3 and is therefore the only column included in the changed row. The inserted rows for Shippers 12, 13, and 14 are batched together under one rs:insert tag. Note that deleted rows may also be batched together, although this is not shown above.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Hierarchical Recordsets in XML

ADO allows persistence of hierarchical **Recordset** objects into XML. With hierarchical **Recordset** objects, the value of a field in the parent **Recordset** is another **Recordset**. Such fields are represented as child elements in the XML stream rather than an attribute. The following example demonstrates this case:

```
Rs.Open "SHAPE {select stor_id, stor_name, state from stores} APPEND
```

The following is the XML format of the persisted **Recordset**:

```
<xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"      xmlns:d
  xmlns:z="#RowsetSchema">
  <s:Schema id="RowsetSchema">
    <s:ElementType name="row" content="eltOnly" rs:updatable="true">
      <s:AttributeType name="stor_id" rs:number="1"
        rs:writeunknown="true">
        <s:datatype dt:type="string" dt:maxLength="4"
          rs:fixedlength="true" rs:maybenull="false"/>
        </s:AttributeType>
      <s:AttributeType name="stor_name" rs:number="2" rs:nullable="t
        rs:writeunknown="true">
        <s:datatype dt:type="string" dt:maxLength="40"/>
        </s:AttributeType>
      <s:AttributeType name="state" rs:number="3" rs:nullable="true"
        rs:writeunknown="true">
        <s:datatype dt:type="string" dt:maxLength="2"
          rs:fixedlength="true"/>
        </s:AttributeType>
      <s:ElementType name="rsSales" content="eltOnly"
        rs:updatable="true" rs:relation="010000000100000000000000">
        <s:AttributeType name="stor_id" rs:number="1"
          rs:writeunknown="true">
          <s:datatype dt:type="string" dt:maxLength="4"
            rs:fixedlength="true" rs:maybenull="false"/>
          </s:AttributeType>
        <s:AttributeType name="ord_num" rs:number="2"
          rs:writeunknown="true">
          <s:datatype dt:type="string" dt:maxLength="20"
            rs:maybenull="false"/>
          </s:AttributeType>
        <s:AttributeType name="ord_date" rs:number="3"
          rs:writeunknown="true">
          <s:datatype dt:type="dateTime" dt:maxLength="16"
            rs:scale="3" rs:precision="23" rs:fixedlength="true"
```

```

        rs:maybenull="false"/>
    </s:AttributeType>
    <s:AttributeType name="qty" rs:number="4" rs:writeunknown="t
        <s:datatype dt:type="i2" dt:maxLength="2" rs:precision="5"
            rs:fixedlength="true" rs:maybenull="false"/>
        </s:AttributeType>
        <s:extends type="rs:rowbase"/>
    </s:ElementType>
    <s:extends type="rs:rowbase"/>
</s:ElementType>
</s:Schema>
<rs:data>
    <z:row stor_id="6380" stor_name="Eric the Read Books" state="WA"
        <rsSales stor_id="6380" ord_num="6871"
            ord_date="1994-09-14T00:00:00" qty="5"/>
        <rsSales stor_id="6380" ord_num="722a"
            ord_date="1994-09-13T00:00:00" qty="3"/>
    </z:row>
    <z:row stor_id="7066" stor_name="Barnum's" state="CA">
        <rsSales stor_id="7066" ord_num="A2976"
            ord_date="1993-05-24T00:00:00" qty="50"/>
        <rsSales stor_id="7066" ord_num="QA7442.3"
            ord_date="1994-09-13T00:00:00" qty="75"/>
    </z:row>
    <z:row stor_id="7067" stor_name="News & Brews" state="CA">
        <rsSales stor_id="7067" ord_num="D4482"
            ord_date="1994-09-14T00:00:00" qty="10"/>
        <rsSales stor_id="7067" ord_num="P2121"
            ord_date="1992-06-15T00:00:00" qty="40"/>
        <rsSales stor_id="7067" ord_num="P2121"
            ord_date="1992-06-15T00:00:00" qty="20"/>
        <rsSales stor_id="7067" ord_num="P2121"
            ord_date="1992-06-15T00:00:00" qty="20"/>
    </z:row>
    ...
</rs:data>
</xml>

```

The exact order of the columns in the parent **Recordset** is not obvious when it is persisted in this manner. Any field in the parent may contain a child **Recordset**. The Persistence Provider persists out all scalar columns first as attributes and then persists out all child **Recordset** "columns" as child elements of the parent row. The ordinal position of the field in the parent **Recordset** can be obtained by looking at the schema definition of the **Recordset**. Every field has an OLE DB property, **rs:number**, defined in the **Recordset** schema namespace that contains the ordinal number for that field.

The names of all fields in the child **Recordset** are concatenated with the name of the field in the parent **Recordset** that contains this child. This is to ensure that there are no name collisions in cases where parent and child **Recordsets** both contain a field that is obtained from two different tables but is named singularly.

When saving hierarchical **Recordsets** into XML, you should be aware of the following restrictions in ADO:

- A hierarchical **Recordset** with pending updates cannot be persisted into XML.
- A hierarchical **Recordset** created with a parameterized shape command cannot be persisted (in either XML or ADTG format).
- ADO currently saves the relationship between the parent and the child **Recordsets** as a binary large object (BLOB). XML tags to describe this relationship have not yet been defined in the rowset schema namespace.
- When a hierarchical **Recordset** is saved, all child **Recordsets** are saved along with it. If the current **Recordset** is a child of another **Recordset**, its parent is not saved. All child **Recordsets** that form the subtree of the current **Recordset** are saved.

When a hierarchical **Recordset** is reopened from its XML-persisted format, you must be aware of the following limitations:

- If the child record contains records for which there are no corresponding parent records, these rows are not written out in the XML representation of the hierarchical **Recordset**. Thus, these rows will be lost when the **Recordset** is reopened from its persisted location.
- If a child record has references to more than one parent record, then on reopening the **Recordset**, the child **Recordset** may contain duplicate records. However, these duplicates will only be visible if the user works directly with the underlying child rowset. If a chapter is used to navigate the child **Recordset** (that is the only way to navigate through ADO), the duplicates are not visible.

ADO 2.5 

Recordset Dynamic Properties in XML

The following **Recordset** provider-specific properties (from the Client Cursor Engine) are currently persisted into the XML format:

- **Update Resync**
- **Unique Table**
- **Unique Schema**
- **Unique Catalog**
- **Resync Command**
- **IRowsetChange**
- **24IRowsetUpdate**
- **CommandTimeout**
- **BatchSize**
- **UpdateCriteria**
- **Reshape Name**
- **AutoRecalc**

These properties are saved in the schema section as attributes of the element definition for the **Recordset** being persisted. These attributes are defined in the rowset schema namespace and must have the prefix "rs:".

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

XSLT Transformations

XSLT can be applied to the generated XML to transform it into another format. Understanding the XML format in ADO helps in developing XSLT templates that can transform it into a more user-friendly form.

For example, you know that each row of the **Recordset** is saved as the `z:row` element inside the `rs:data` element. Similarly, each field of the **Recordset** is saved as an attribute-value pair for this element.

The following XSLT script can be applied to the XML shown in the previous section to transform it into an HTML table to be displayed in the browser:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<body STYLE="font-family:Arial, helvetica, sans-serif; font-size:12p
<table border="1" style="table-layout:fixed" width="600">
  <col width="200"></col>
  <tr bgcolor="teal">
    <th><font color="white">CustomerId</font></th>
    <th><font color="white">CompanyName</font></th>
    <th><font color="white">ContactName</font></th>
  </tr>
<xsl:for-each select="xml/rs:data/z:row">
  <tr bgcolor="navy">
    <td><font color="white"><xsl:value-of select="@CustomerID"/></font>
    <td><font color="white"><xsl:value-of select="@CompanyName"/></font>
    <td><font color="white"><xsl:value-of select="@ContactName"/></font>
  </tr>
</xsl:for-each>
</table>
</body>
</html>
```

The XSLT converts the XML stream generated by the ADO **Save** method into an HTML table which displays each field of the **Recordset** along with a table heading. Table headings and rows also are assigned different fonts and colors.

ADO 2.5 

Saving to the XML DOM Object

You can save a **Recordset** in XML format to an instance of an MSXML DOM object, as shown in the following Visual Basic code:

```
Dim xDOM As New MSXML.DOMDocument
Dim rsXML As New ADO.DB.Recordset
Dim sSQL As String, sConn As String

sSQL = "SELECT customerid, companyname, contactname FROM customers"
sConn="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\Program Files
      "\Common Files\System\msadc\samples\NWind.mdb"
rsXML.Open sSQL, sConn
rsXML.Save xDOM, adPersistADO 'Save Recordset directly into a DOM
...
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

XML Security Considerations

The ADO **Save** and **Open** methods on the **Recordset** object are not considered safe operations to run in Internet Explorer. Thus, if these methods are used in a script code that is running in an application or control that is hosted in a browser, the security configuration of the browser will have an effect on its behavior.

Internet Explorer 5 provides security restrictions for such operations by default in the Internet zones. Under that configuration, the **Recordset** cannot make any access to the local file system on the client or access any data sources outside the domain of the server from which the page has been downloaded. Specifically, when running inside the browser host, a **Recordset** can be saved back to a file only if it is on the same server from which the page was downloaded. Similarly, you can open a **Recordset** by loading it from a file only if that file is on the same server from which the page was downloaded.

For more information about security in Internet Explorer, see the technical article "ADO and RDS Security Issues in Microsoft Internet Explorer."

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

XML Recordset Persistence Scenario

In this scenario, you will create an Active Server Pages (ASP) application that saves the contents of a **Recordset** object directly to the ASP **Response** object.

Note This scenario requires that your server have Internet Information Server 5.0 (IIS) or later installed.

The returned **Recordset** is displayed in Internet Explorer using an [RDS.DataControl](#).

The following steps are necessary to create this scenario:

1. Set up the application.
2. Get the data.
3. Send the data.
4. Receive and display the data.

Next [Step 1: Set Up the Application](#)

See Also

[Save Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Step 1: Set Up the Application

Create an IIS virtual directory named "XMLPersist" with script permissions. Create two new text files in the folder to which the virtual directory points, one named "XMLResponse.asp," the other named "Default.htm."

Next [Step 2: Get the Data](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

Step 2: Get the Data

In this step, you will write the code to open an ADO **Recordset** and prepare to send it to the client. Open the file XMLResponse.asp with a text editor, such as Windows Notepad, and insert the following code:

```
<%@ language="VBScript" %>

<!-- #include file='adovbs.inc' -->

<%
    Dim strSQL, strCon
    Dim adoRec
    Dim adoCon
    Dim xmlDoc

    ' You will need to change "slqServer" below to the name of the SQL
    ' server machine to which you want to connect.
    strCon = "Provider=sqloledb;Data Source=sqlServer;Initial Catalog=
Set adoCon = server.createObject("ADODB.Connection")
adoCon.Open strCon

    strSQL = "SELECT Title, Price FROM Titles ORDER BY Price"
    Set adoRec = Server.CreateObject("ADODB.Recordset")
    adoRec.Open strSQL, adoCon, adOpenStatic, adLockOptimistic, adCmdT
```

Be sure to change the value of the Data Source parameter in strCon to the name of your Microsoft SQL Server computer.

Keep the file open and go on to the next step.

Next [Step 3: Send the Data](#)

Step 3: Send the Data

Now that you have a **Recordset**, you need to send it to the client by saving it as XML to the ASP **Response** object. Add the following code to the bottom of XMLResponse.asp:

```
Response.ContentType = "text/xml"
Response.Expires = 0
Response.Buffer = False

Response.Write "<?xml version='1.0'?>" & vbNewLine
adoRec.save Response, adPersistXML
adoRec.Close
Set adoRec=Nothing
%>
```

Notice that the ASP **Response** object is specified as the destination for the **Recordset** [Save](#) method. The destination of the **Save** method can be any object that supports the **IStream** interface, such as an ADO [Stream](#) object, or a file name that includes the complete path to which the **Recordset** is to be saved.

Save and close XMLResponse.asp before going to the next step. Also copy the adovbs.inc file from C:\Program Files\Common Files\System\Ado folder to the same folder where you have the XMLResponse.asp file.

Next [Step 4: Receive the Data](#)

Step 4: Receive and Display the Data

In this step you will create an HTML file with an embedded [RDS.DataControl](#) object that points at the XMLResponse.asp file to get the **Recordset**. Open default.htm with a text editor, such as Windows Notepad, and add the code below. Replace "sqlserver" in the URL with the name of your server computer.

```
<HTML>
<HEAD><TITLE>ADO Recordset Persistence Sample</TITLE></HEAD>
<BODY>

<TABLE DATASRC="#RDC1" border="1">
  <TR>
<TD><SPAN DATAFLD="title"></SPAN></TD>
<TD><SPAN DATAFLD="price"></SPAN></TD>
  </TR>
</TABLE>
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID="RDC
  <PARAM NAME="URL" VALUE="XMLResponse.asp">
</OBJECT>

</BODY>
</HTML>
```

Close the default.htm file and save it to the same folder where you saved XMLResponse.asp. Using Internet Explorer 4.0 or later, open the URL <http://sqlserver/XMLPersist/default.htm> and observe the results. The data is displayed in a bound DHTML table. Now open the URL <http://sqlserver/XMLPersist/XMLResponse.asp> and observe the results. The XML is displayed.

ADO 2.5 

Chapter 6: Error Handling

ADO uses several different methods to notify an application of errors that occur. This chapter discusses the types of errors that can occur when you are using ADO and how your application is notified. It concludes by making suggestions about how to handle those errors.

How Does ADO Report Errors?

ADO notifies you about errors in several ways:

- ADO errors generate a run-time error. Handle an ADO error the same way you would any other run-time error, such as using an **On Error** statement in Visual Basic.
- Your program can receive errors from OLE DB. An OLE DB error generates a run-time error as well.
- If the error is specific to your data provider, one or more **Error** objects are placed in the **Errors** collection of the **Connection** object that was used to access the data store when the error occurred.
- If the process that raised an event also produced an error, error information is placed in an **Error** object and passed as a parameter to the event. See [Chapter 7: Handling ADO Events](#) for more information about events.
- Problems that occur when processing batch updates or other bulk operations involving a **Recordset** can be indicated by the **Status** property of the **Recordset**. For example, schema constraint violations or insufficient permissions can be specified by **RecordStatusEnum** values.
- Problems that occur involving a particular **Field** in the current record are also indicated by the **Status** property of each **Field** in the **Fields** collection of the **Record** or **Recordset**. For example, updates that could not be completed or incompatible data types can be specified by **FieldStatusEnum** values.

The following sections describe each of these notification methods in more detail.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

ADO Errors

ADO Errors are reported to your program as run-time errors. You can use the error-trapping mechanism of your programming language to trap and handle them. For example, in Visual Basic, use the **On Error** statement. In Visual J++, use a **try-catch** block. In Visual C++, it depends on the method you are using to access the ADO libraries. With #import, use a **try-catch** block. Otherwise, C++ programmers need to explicitly retrieve the error object by calling **GetErrorInfo**. The following Visual Basic sub procedure demonstrates trapping an ADO error:

```
' BeginErrorHandlingVB01
Private Sub Form_Load()
    ' Turn on error handling
    On Error GoTo FormLoadError

    'Open the database and the recordset for processing.
    '
    Dim strCnn As String
    strCnn = "Provider='sqloledb';" & _
        "Data Source='MySqlServer';" & _
        "Initial Catalog='Northwind';Integrated Security='SSPI';"

    ' cnn is a Public Connection Object because
    ' it was defined WithEvents
    Set cnn = New ADODB.Connection
    cnn.Open strCnn

    ' The next line of code intentionally causes
    ' an error by trying to open a connection
    ' that has already been opened.
    cnn.Open strCnn

    ' rst is a Public Recordset because it
    ' was defined WithEvents
    Set rst = New ADODB.Recordset
    rst.Open "Customers", cnn

    Exit Sub

' Error handler
FormLoadError:
    Dim strErr As String
    Select Case Err
```

```

    Case adErrObjectOpen
        strErr = "Error #" & Err.Number & ": " & Err.Description
        strErr = strErr & "Error reported by: " & Err.Source & v
        strErr = strErr & "Help File: " & Err.HelpFile & vbCrLf
        strErr = strErr & "Topic ID: " & Err.HelpContext
        MsgBox strErr
        Debug.Print strErr
        Err.Clear
        Resume Next
    ' If some other error occurs that
    ' has nothing to do with ADO, show
    ' the number and description and exit.
    Case Else
        strErr = "Error #" & Err.Number & ": " & Err.Description
        MsgBox strErr
        Debug.Print strErr
        Unload Me
    End Select
End Sub
' EndErrorHandlingVB01

```

This **Form_Load** event procedure intentionally creates an error by trying to open the same **Connection** object twice. The second time the **Open** method is called, the error handler is activated. In this case the error is of type **adErrObjectOpen**, so the error handler displays the following message before resuming program execution:

```

Error #3705: Operation is not allowed when the object is open.
Error reported by: ADODB.Connection
Help File: E:\WINNT\HELP\AD0260.CHM Topic ID: 1003705

```

The error message includes each piece of information provided by the Visual Basic **Err** object except for the **LastDLLError** value, which does not apply here. The error number tells you which error has occurred. The description is useful in cases in which you do not want to handle the error yourself. You can simply pass it along to the user. Although you will usually want to use messages customized for your application, you cannot anticipate every error; the description gives some clue as to what went wrong. In the sample code, the error was reported by the **Connection** object. You will see the object's type or programmatic ID here—not a variable name.

Note The Visual Basic **Err** object only contains information about the most recent error. The ADO **Errors** collection of the **Connection** object contains one **Error** object for each error raised by the most recent ADO

operation. Use the **Errors** collection rather than the **Err** object to handle multiple errors. For more information about the **Errors** collection, see [Provider Errors](#). However, if there is no valid **Connection** object, the **Err** object is the only source for information about ADO errors.

What kinds of operations are likely to cause ADO errors? Common ADO errors can involve opening an object such as a **Connection** or **Recordset**, attempting to update data, or calling a method or property that is not supported by your provider.

OLE DB errors can also be passed to your application as run-time errors in the **Errors** collection. For more information about OLE DB error numbers, see Chapter 16 of the OLE DB Programmer's Reference.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

ADO Error Reference

The **ErrorValueEnum** constant describes the ADO error values. For a complete listing of these enumerated constants, including values, see [Appendix B: ADO Errors](#). This section will examine some of the more interesting errors and explain some specific situations that can raise them, or solutions to fix the problem. Both the **ErrorValueEnum** constant and the short positive decimal number are listed.

Number	ErrorValueEnum constant	Description/Possible cause
3000	adErrProviderFailed	Provider failed to perform the requested operation. Arguments are of the wrong type, out of acceptable range, or are in conflict with one another. This is often caused by a typographical error in an SQL SELECT statement. For example, a misspelled field name or table name can generate this error. This error can also occur when a column or table named in a SELECT statement does not exist in the database store.
3001	adErrInvalidArgument	File could not be opened. A misspelled file name was specified, or a file has been moved, renamed, or deleted. Over a network, the drive might be temporarily unavailable, or network traffic might be preventing connection.
3002	adErrOpeningFile	File could not be read. The name of the file is specified incorrectly, the file might have been moved or deleted, or the file might have been corrupted.
3003	adErrReadFile	Write to file failed. You might have

3004 adErrWriteFile

closed a file and then tried to write to it, or the file might be corrupted. If the file is located on a network drive, transient network conditions might prevent writing to a network drive. Either **BOF** or **EOF** is True, or the current record has been deleted. Requested operation requires a current record.

3021 adErrNoCurrentRecord

An attempt was made to update records by using **Find** or **Seek** to move the record pointer to the desired record. If the record is not found, **EOF** will be True. This error can occur after a failed **AddNew** or **Delete** because there is no current record when these methods fail.

3219 adErrIllegalOperation

Operation is not allowed in this context.

3220 adErrCantChangeProvider

Supplied provider is different from the one already in use.

3246 adErrInTransaction

Connection object cannot be explicitly closed while in a transaction. A **Recordset** or **Connection** object that is currently participating in a transaction cannot be closed. Call either **RollbackTrans** or **CommitTrans** before closing object.

3251 adErrFeatureNotAvailable

The object or provider is not capable of performing the requested operation. Some operations depend on a particular provider version.

3265 adErrItemNotFound

Item cannot be found in the collection corresponding to the requested record ID or ordinal. An incorrect field or

3367	adErrObjectInCollection	name has been specified. Object is already in collection. Cannot append. An object cannot be added to the same collection twice.
3420	adErrObjectNotSet	Object is no longer valid. Application uses a value of the <code>value</code> type for the current operation. You might have supplied a string to a operation that expects a stream, for example.
3421	adErrDataConversion	Operation is not allowed when the object is closed. The ConnectionRecordset has been closed. For example, some other routine might have closed a global object. You can prevent this error by checking the State property before you attempt the operation.
3704	adErrObjectClosed	Operation is not allowed when the object is open. An object that is already open cannot be opened. Fields cannot be appended to an open Recordset . Provider cannot be found. It may not be properly installed.
3705	adErrObjectOpen	The name of the provider might be incorrectly specified, the specific provider might not be installed on the computer where your code is being executed, or the installation might have become corrupted.
3706	adErrProviderNotFound	The ActiveConnection property of the Recordset object, which has a Command object as its source, cannot be changed. The application attempted to assign a new Connection object to a Recordset .
3707	adErrBoundToCommand	

		that has a Command object as its source.
3708	adErrInvalidParamInfo	Parameter object is improperly defined. Inconsistent or incomplete information was provided.
3709	adErrInvalidConnection	The connection cannot be used to perform this operation. It is either closed or invalid in this context.
3710	adErrNotReentrant	Operation cannot be performed during a processing event. An operation cannot be performed within an event handler that causes the event to occur again. For example, navigation methods should not be called from within a WillMove event handler.
3711	adErrStillExecuting	Operation cannot be performed while executing asynchronously.
3712	adErrOperationCancelled	Operation has been canceled by user. The application has called CancelUpdate or CancelBatch method and the current operation has been canceled.
3713	adErrStillConnecting	Operation cannot be performed while connecting asynchronously.
3714	adErrInvalidTransaction	Coordinating transaction is invalid because it has not started.
3715	adErrNotExecuting	Operation cannot be performed while not executing.
3716	adErrUnsafeOperation	Safety settings on this computer prohibit accessing a data source on another domain.
3717	adWrnSecurityDialog	For internal use only. Don't use. (Entry was included for the sake of completeness. This error should appear in your code.)
		For internal use only. Don't use. (Entry included for the sake of

3718	adWrnSecurityDialogHeader	completeness. This error should appear in your code.)
3719	adErrIntegrityViolation	Data value conflicts with the integrity constraints of the field. A new value for a Field would cause a duplicate key. A value that forms one side of a relationship between two records might not be updatable.
3720	adErrPermissionDenied	Insufficient permission prevents writing to the field. The user name in the connection string does not have the proper permissions to write to the Field .
3721	adErrDataOverflow	Data value is too large to be represented by the field data type. A numeric value that is too large for the intended field was assigned. For example, a long integer value was assigned to a short integer field.
3722	adErrSchemaViolation	Data value conflicts with the data type or constraints of the field. The data store has validation constraints that differ from the Field value.
3723	adErrSignMismatch	Conversion failed because the data value was signed and the field data type used by the provider was unsigned.
3724	adErrCantConvertvalue	Data value cannot be converted for reasons other than sign mismatch or data overflow. For example, conversion would have truncated data.
3725	adErrCantCreate	Data value cannot be set or retrieved because the field data type was unknown, or the provider had insufficient resources to perform the operation.

3726 adErrColumnNotOnThisRow

Record does not contain this field
incorrect field name was specified
a field not in the **Fields** collection
the current record was referenced
Either the source URL or the part
of the destination URL does not
There is a typographical error in
either the source or destination URL
You might have

3727 adErrURLDoesNotExist

`http://mysite/photo/myphoto`
when you should actually have
`http://mysite/photos/myphoto`
instead. The typographical error
parent URL (in this case, *photo*
instead of *photos*) has caused the
error.

3728 adErrTreePermissionDenied

Permissions are insufficient to a
tree or subtree. The user named
connection string does not have
appropriate permissions.

3729 adErrInvalidURL

URL contains invalid characters
Make sure the URL is typed correctly.
The URL follows the scheme
registered to the current provider.
example, Internet Publishing Protocol
is registered for http).

3730 adErrResourceLocked

Object represented by the specified
URL is locked by one or more other
processes. Wait until the process
finished and attempt the operation
again. The object you are trying
access has been locked by another
user or by another process in your
application. This is most likely to
arise in a multi-user environment.
Copy operation cannot be performed.
Object named by destination URL
already exists. Specify

3731	adErrResourceExists	<p>adCopyOverwrite to replace the object. If you do not specify adCopyOverwrite when copying files in a directory, the copy fails when you try to copy an item that already exists in the destination location.</p>
3732	adErrCannotComplete	<p>The server cannot complete the operation. This might be because the server is busy with other operations or it might be low on resources.</p>
3733	adErrVolumeNotFound	<p>Provider cannot locate the storage device indicated by the URL. Make sure the URL is typed correctly. The URL of the storage device might be incorrect, but this error can occur for other reasons. The device might be offline or a large volume of network traffic might prevent the connection from being made.</p>
3734	adErrOutOfSpace	<p>Operation cannot be performed. Provider cannot obtain enough storage space. There might not be enough RAM or hard-drive space for temporary files on the server.</p>
3735	adErrResourceOutOfScope	<p>Source or destination URL is outside the scope of the current record.</p>
3736	adErrUnavailable	<p>Operation failed to complete and status is unavailable. The field is unavailable or the operation was attempted. Another user might have changed or deleted the field you are trying to access.</p>
3737	adErrURLNamedRowDoesNotExist	<p>Record named by this URL does not exist. While attempting to open a record using a Record object, either the name or the path to the file was misspelled.</p>

3738	adErrDelResOutOfScope	The URL of the object to be deleted is outside the scope of the current record.
3747	adErrCatalogNotSet	Operation requires a valid ParentCatalog .
3748	adErrCantChangeConnection	Connection was denied. The new connection you requested has different characteristics than the one already in use.
3749	adErrFieldsUpdateFailed	Fields update failed. For further information, examine the Status property of individual field objects. This error can occur in two situations: when changing a Field object's values in the process of changing or adding a record to the database; and when changing the properties of the Field object itself.
3750	adErrDenyNotSupported	The Record or Recordset update failed due to a problem with one of the fields in the current record. Enumerate the Fields collection and check the Status property of each field to determine the cause of the problem.
3751	adErrDenyTypeNotSupported	Provider does not support sharing restrictions. An attempt was made to restrict file sharing and your provider does not support the concept. Provider does not support the requested kind of sharing restriction. An attempt was made to establish a particular type of file-sharing restriction that is not supported by your provider. See the provider's documentation to determine why.

sharing restrictions are supported

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Provider Errors

When a provider error occurs, a run-time error of -2147467259 is returned. When you receive this error, check the active **Connection** object's **Errors** collection, which will contain one or more errors describing what happened.

The ADO Errors Collection

Because a particular ADO operation can produce multiple provider errors, ADO exposes a collection of error objects through the **Connection** object. This collection contains no objects if an operation concludes successfully and contains one or more **Error** objects if something went wrong and the provider raised one or more errors. Examine each individual error object in order to determine the exact cause of the error.

Once you have finished handling any errors that have occurred, you can clear the collection by calling the **Clear** method. It is particularly important to explicitly clear the **Errors** collection before you call the **Resync**, **UpdateBatch**, or **CancelBatch** method on a **Recordset** object, the **Open** method on a **Connection** object, or set the **Filter** property on a **Recordset** object. By clearing the collection explicitly, you can be certain that any **Error** objects in the collection are not left over from a previous operation.

Some operations can generate warnings as well as errors. Warnings are also represented by **Error** objects in the **Errors** collection. When a provider adds a warning to the collection, it does not generate a run-time error. Check the **Count** property of the **Errors** collection to determine if a warning was produced by a particular operation. If the count is one or greater, an **Error** object has been added to the collection. Once you have determined that the **Errors** collection contains errors or warnings, you can iterate through the collection and retrieve information about each of the **Error** objects it contains. The following short Visual Basic example demonstrates this:

```
' BeginErrorHandlingVB02
Private Function DeleteCustomer(ByVal CompanyName As String) As Long
    On Error GoTo DeleteCustomerError

    rst.Find "CompanyName='" & CompanyName & "'"

DeleteCustomerError:

    Dim objError As ADODB.Error
    Dim strError As String

    If cnn.Errors.Count > 0 Then
        For Each objError In cnn.Errors
            strError = strError & "Error #" & objError.Number & _
```

```

        " " & objError.Description & vbCrLf & _
        "NativeError: " & objError.NativeError & vbCrLf & _
        "SQLState: " & objError.SQLState & vbCrLf & _
        "Reported by: " & objError.Source & vbCrLf & _
        "Help file: " & objError.HelpFile & vbCrLf & _
        "Help Context ID: " & objError.HelpContext
    Next
    MsgBox strError
End If
End Function
' EndErrorHandlingVB02

```

The error-handling routine includes a **For Each** loop that examines each object in the **Errors** collection. In this example, it simply accumulates a message for display. In a working program, you would write code to perform an appropriate task for each error, such as closing all open files and shutting down the program in an orderly fashion.

The Error Object

By examining an **Error** object you can determine what error occurred, and more importantly, what application or what object caused the error. The **Error** object has the following properties:

Property name	Description
Description	A text description of the error that occurred.
HelpContext, HelpFile	Refers to the help topic and help file that contain a description of the error that occurred.
NativeError	The provider-specific error number.
Number	A Long Integer that represents the number (listed in the ErrorValueEnum) of the error that occurred.
Source	Indicates the name of the object or application that generated an error.
SQLState	A five-character error code that the provider returns during the process of a SQL statement.

The ADO **Error** object is quite similar to the standard Visual Basic **Err** object. Its properties describe the error that occurred. In addition to the number of the error, you also receive two related pieces of information. The **NativeError** property contains an error number specific to the provider you are using. In the previous example, the provider is the Microsoft OLE DB Provider for SQL Server, so **NativeError** will contain errors specific to SQL Server. The **SQLState** property has a five-letter code that describes an error in a SQL statement.

Event-Related Errors

The **Error** object is also used when event-related errors occur. You can determine if an error occurred in the process that raised an ADO event by checking the **Error** object passed as an event parameter.

If the operation that causes an event is concluded successfully, the *adStatus* parameter of the event handler will be set to *adStatusOK*. On the other hand, if the operation that raised the event was unsuccessful, the *adStatus* parameter is set to *adStatusErrorsOccurred*. In that case, the *pError* parameter will contain an **Error** object that describes the error.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Field-Related Error Information

If an error is directly related to a field—for example, if the data is missing or if it is the wrong type for the field—you can retrieve more information about the cause of the problem by examining the **Field** object's **Status** property. This property has been enhanced to provide specific information about the problem. So, for example, when a call to **UpdateBatch** fails, the cause of the problem can be determined by examining the **Status** property of the **Fields** in each of the effected records. The property will contain one of the values in the **FieldStatusEnum** constant. The following table includes those values that are of particular interest when an error occurs.

Constant	Value	Description
adFieldCantConvertValue	2	Indicates that the field cannot be retrieved or stored without loss of data.
adFieldDataOverflow	6	Indicates that the data returned from the provider overflowed the data type of the field.
adFieldDefault	13	Indicates that the default value for the field was used when setting data.
adFieldIgnore	15	Indicates that this field was skipped when setting data values in the source. No value was set by the provider.
adFieldIntegrityViolation	10	Indicates that the field cannot be modified because it is a calculated or derived entity.
adFieldIsNull	3	Indicates that the provider returned a null value.
adFieldOutOfSpace	22	Indicates that the provider is unable to obtain enough storage space to complete a move or copy operation.

ADO 2.5 

Recordset-Related Error Information

During batch processing, the **Status** property of the **Recordset** object provides information about the individual records in the **Recordset**. Before a batch update takes place, the **Status** property of the **Recordset** reflects information about records to be added, changed and deleted. After **UpdateBatch** has been called, the **Status** property indicates the success or failure of the operation. As you move from record to record in the **Recordset**, the value of the **Status** property changes to describe the status of the current record.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Handling Errors in other Languages

So far, this chapter has discussed errors from a Microsoft Visual Basic point of view. The remainder of this section will give you an overview of error handling in languages other than Visual Basic.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Handling Errors in VBScript

There is little difference between the methods used in Visual Basic and those used with VBScript. The primary difference is that VBScript does not support the concept of error handling by continuing execution at a label. In other words, you cannot use `On Error GoTo` in VBScript. Instead, use `On Error Resume Next` and then check both **Err.Number** and the **Count** property of the **Errors** collection, as shown in the following example:

```
<!-- BeginErrorExampleVBS -->
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>Error Handling Example (VBScript)</TITLE>
</HEAD>
<BODY>

<h1>Error Handling Example (VBScript)</h1>

<%
    Dim errLoop
    Dim strError

    On Error Resume Next

    ' Intentionally trigger an error.
    Set cnn1 = Server.CreateObject("ADODB.Connection")
    cnn1.Open "nothing"

    If cnn1.Errors.Count > 0 Then
        ' Enumerate Errors collection and display
        ' properties of each Error object.
        For Each errLoop In cnn1.Errors
            strError = "Error #" & errLoop.Number & "<br>" & _
                " " & errLoop.Description & "<br>" & _
                " (Source: " & errLoop.Source & ")" & "<br>" & _
                " (SQL State: " & errLoop.SQLState & ")" & "<br>" & _
                " (NativeError: " & errLoop.NativeError & ")" & "<br>"
            If errLoop.HelpFile = "" Then
                strError = strError & _
                    " No Help file available" & _
                    "<br><br>"
            Else
                strError = strError & _
```

```
        " (HelpFile: " & errLoop.HelpFile & ")" & "<br>"
        " (HelpContext: " & errLoop.HelpContext & ")"
        "<br><br>"
    End If
    Response.Write ("<p>" & strError & "</p>")
Next
End If
%>

</BODY>
</HTML>
<!-- EndErrorExampleVBS -->
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Handling Errors in Visual C++

In COM, most operations return an HRESULT return code that indicates whether a function completed successfully. The #import directive generates wrapper code around each "raw" method or property and checks the returned HRESULT. If the HRESULT indicates failure, the wrapper code throws a COM error by calling `_com_issue_errorex()` with the HRESULT return code as an argument. COM error objects can be caught in a **try-catch** block. (For efficiency's sake, catch a reference to a `_com_error` object.)

Remember, these are ADO errors: they result from the ADO operation failing. Errors returned by the underlying provider appear as **Error** objects in the **Connection** object's **Errors** collection.

The #import directive only creates error-handling routines for methods and properties declared in the ADO .dll. However, you can take advantage of this same error-handling mechanism by writing your own error-checking macro or inline function. See the topic Visual C++ Extensions for examples.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Handling Errors in Visual J++

Handle ADO errors in your Microsoft Visual J++ applications using a **try catch** block. Once an error has been thrown, you can iterate through the collection, successively handling each error. The following Visual J++ example shows a console application that deliberately causes an error.

When the **catch** block is activated, it calls the `PrintProviderError` function to display the errors. The `PrintProviderError` function iterates through the **Errors** collection and sends a line to the standard output device that describes each error in the collection.

```
// BeginErrorExampleVJ
/**
 * This class can take a variable number of parameters on the command
 * line. Program execution begins with the main() method. The class
 * constructor is not invoked unless an object of type 'Class1'
 * created in the main() method.
 */

import com.ms.wfc.data.*;
import java.io.* ;

public class ErrorExample
{
    /**
     * The main entry point for the application.
     *
     * @param args Array of parameters passed to the application
     * via the command line.
     */
    public static void main (String[] args)
    {
        DescriptionX();
        System.exit(0);
    }

    static void DescriptionX()
    {
        BufferedReader in = new
        BufferedReader(new InputStreamReader(System.in));

        // Define ADO Objects.
```

```

Connection cnConn1 = null;

try
{
    // Create an error by trying to
    // Open a database that doesn't exist.
    cnConn1 = new Connection();
    cnConn1.open("nothing");
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.
    PrintProviderError(cnConn1);
}

try
{
    System.out.println("\nPress <Enter> key to continue.");
    in.readLine();
}
// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}
}

// PrintProviderError Function
static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors colle
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.get
                + "\t" + ErrItem.getDescription() );
        }
    }
}
}

```

```
// PrintIOError Function
static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n");
}
}
// EndErrorExampleVJ
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Handling Errors in JScript

Your Microsoft JScript code must check the **Count** property of the **Connection** object's **Errors** collection. If the value is greater than 0, iterate through the collection and print the values as you would in any of the other languages.

```
<!-- BeginErrorExampleJS -->
<%@ Language=JScript %>
<HTML>
<HEAD>
<title>Error Handling Example (JScript)</title>
</HEAD>
<BODY>
<h1>Error Handling Example (JScript)</h1>
<%
    var cnn1 = Server.CreateObject("ADODB.Connection");
    var errLoop = Server.CreateObject("ADODB.Error");
    var strError = new String;

    try
    {
        // Intentionally trigger an error.
        cnn1.Open("nothing");
    }
    catch(e)
    {
        Response.Write(e.message);
    }
%>

</BODY>
</HTML>
<!-- EndErrorExampleJS -->
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Anticipating Errors

Error prevention is at least as important as error handling. This final section contains a short list of precautions your application can take to help make errors less likely to occur.

Check the state of objects by checking the value in the **State** property before trying to perform an operation using those objects. For example, if your application uses a global **Connection**, check its **State** property to see if it is already open before calling the **Open** method.

- Any program that accepts data from a user must include code to validate that data before sending it to the data store. You cannot rely on the data store, the provider, ADO, or even your programming language to notify you of problems. You must check every byte entered by your users, making sure that data is the correct type for its field and that required fields are not empty.

Check the data before you try to write any data to the data store. The easiest way to do so is to handle the **WillMove** event or the **WillUpdateRecordset** event. For a more complete discussion of handling ADO events, see [Chapter 7: Handling ADO Events](#).

Make sure that **Recordset** objects are not beyond the boundaries of the **Recordset** before attempting to move the record pointer. If you try to **MoveNext** when **EOF** is True or **MovePrev** when **BOF** is True, an error will occur. If you perform any of the **Move** methods when both **EOF** and **BOF** are True, an error will be generated.

Errors also will occur if you try to perform operations such as **Seek** and **Find** on an empty **Recordset**.

ADO 2.5 

Chapter 7: Handling ADO Events

The ADO event model supports certain [synchronous](#) and [asynchronous](#) ADO operations that issue *events*, or notifications, before the operation starts or after it completes. An event is actually a call to an event-handler routine that you define in your application.

If you provide handler functions or procedures for the group of events that occur before the operation starts, you can examine or modify the parameters that were passed to the operation. Because it has not been executed yet, you can either cancel the operation or allow it to complete.

The group of events that occur after an operation completes are especially important if you use ADO asynchronously. For example, an application that starts an asynchronous [Recordset.Open](#) operation is notified by an execution complete event when the operation concludes.

Using the ADO event model adds some overhead to your application but provides far more flexibility than other methods of dealing with asynchronous operations, such as monitoring the [State](#) property of an object with a loop.

See Also

[ADO Event Handler Summary](#) | [ADO Event Instantiation by Language](#) | [ADO Events](#) | [Event Parameters](#) | [Types of Events](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

ADO Event Handler Summary

Two ADO objects can raise events: the [Connection](#) object and the [Recordset](#) object. The **ConnectionEvent** family pertains to operations on the **Connection** object, and the **RecordsetEvent** family pertains to operations on the **Recordset** object.

- **Connection Events:** Events are issued when a transaction on a connection begins, is committed, or is rolled back; when a [Command](#) executes; when a warning occurs during a **Connection Event** operation; or when a **Connection** starts or ends.
- **Recordset Events:** Events are issued around asynchronous fetch operations as well as when you navigate through the rows of a **Recordset** object, change a field in a row of a **Recordset**, change a row in a **Recordset**, open a **Recordset** with a server-side cursor, close a **Recordset**, or make any change whatsoever in the **Recordset**.

The following tables summarize the events and their descriptions.

ConnectionEvent	Description
BeginTransComplete , CommitTransComplete , RollbackTransComplete	Transaction Management —Notification that the current transaction on the connection has started, committed, or rolled back.
WillConnect , ConnectComplete , Disconnect	Connection Management —Notification that the current connection will start, has started, or has ended.
WillExecute , ExecuteComplete	Command Execution Management —Notification that the execution of the current command on the connection will start or has ended.
InfoMessage	Informational —Notification that there is additional information about the current operation.
RecordsetEvent	Description
FetchProgress ,	Retrieval Status —Notification of the progress of a data retrieval operation, or that the retrieval operation has completed. These events are only

[FetchComplete](#)

available if the **Recordset** was opened using a client-side cursor.

[WillChangeField,](#)
[FieldChangeComplete](#)

Field Change Management—Notification that the value of the current field will change, or has changed.

[WillMove, MoveComplete,](#)
[EndOfRecordset](#)

Navigation Management—Notification that the current row position in a **Recordset** will change, has changed, or has reached the end of the **Recordset**.

[WillChangeRecord,](#)
[RecordChangeComplete](#)

Row Change Management—Notification that something in the current row of the **Recordset** will change, or has changed.

[WillChangeRecordset,](#)
[RecordsetChangeComplete](#)

Recordset Change Management—Notification that something in the current **Recordset** will change, or has changed.

See Also

[ADO Event Instantiation by Language](#) | [ADO Events](#) | [Event Parameters](#) | [How Event Handlers Work Together](#) | [Types of Events](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Types of Events

There are two basic types of events. "Will Events," which are called before an operation starts, usually include "Will" in their names—for example, **WillChangeRecordset** or **WillConnect**. Events that are called after an event has been completed usually include "Complete" in their names—for example, **RecordChangeComplete** or **ConnectComplete**. Exceptions exist—such as **InfoMessage**—but these occur after the associated operation has completed.

Will Events

Event handlers called before the operation starts offer you the opportunity to examine or modify the operation parameters, and then either cancel the operation or allow it to complete. These event-handler routines usually have names of the form **WillEvent**.

Complete Events

Event handlers called after an operation completes can notify your application that an operation has concluded. Such an event handler is also notified when a Will event handler cancels a pending operation. These event-handler routines usually have names of the form ***EventComplete***.

Will and Complete events are typically used in pairs.

Other Events

The other event handlers—that is, events whose names are not of the form **WillEvent** or **EventComplete**—are called only after an operation completes. These events are **Disconnect**, **EndOfRecordset**, and **InfoMessage**.

See Also

[ADO Event Handler Summary](#) | [ADO Event Instantiation by Language](#) | [Event Parameters](#) | [How Event Handlers Work Together](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Event Parameters

Every event handler has a status parameter that controls the event handler. For Complete events, this parameter is also used to indicate the success or failure of the operation that generated the event. Most Complete events also have an error parameter to provide information about any error that might have occurred, as well as one or more object parameters that refer to the ADO objects used to perform the operation. For example, the [ExecuteComplete](#) event includes object parameters for the **Command**, **Recordset**, and **Connection** objects associated with the event. In the following Microsoft Visual Basic example, you can see the pCommand, pRecordset and pConnection objects which represent the **Command**, **Recordset**, and **Connection** objects used by the **Execute** method.

```
Private Sub connEvent_ExecuteComplete(ByVal RecordsAffected As Long,
    ByVal pError As ADODB.Error, _
    adStatus As ADODB.EventStatusEnum, _
    ByVal pCommand As ADODB.Command, _
    ByVal pRecordset As ADODB.Recordset, _
    ByVal pConnection As ADODB.Connection)
```

Except for the **Error** object, the same parameters are passed to the Will events. This gives you the opportunity to examine each of the objects to be used in the pending operation and determine whether the operation should be allowed to complete.

Some event handlers have a *Reason* parameter, which provides additional information about why the event occurred. For example, the **WillMove** and **MoveComplete** events can occur due to any one of the navigation methods (**MoveNext**, **MovePrevious**, and so on) being called or as the result of a query.

Status Parameter

When the event-handler routine is called, the *Status* parameter is set to one of the following values.

Value	Description
adStatusOK	Passed to both Will and Complete events. This value means that the operation that caused the event completed successfully.
adStatusErrorsOccurred	Passed to Complete events only. This value means that the operation that caused the event was unsuccessful, or a Will event canceled the operation. Check the <i>Error</i> parameter for more details.
adStatusCantDeny	Passed to Will events only. This value means that the operation cannot be canceled by the Will event. It must be performed.

If you determine in your Will event that the operation should continue, leave the *Status* parameter unchanged. As long as the incoming status parameter was not set to **adStatusCantDeny**, however, you can cancel the pending operation by changing *Status* to **adStatusCancel**. When you do so, the Complete event associated with the operation has its *Status* parameter set to **adStatusErrorsOccurred**. The **Error** object passed to the Complete event will contain the value **adErrOperationCancelled**.

If you no longer want to process an event, you can set *Status* to **adStatusUnwantedEvent** and your application will no longer receive notification of that event. Remember, however, that some events can be raised for more than one reason. In that case, you must specify **adStatusUnwantedEvent** for each possible reason. For example, in order to stop receiving notification of pending **RecordChange** events, you must set the *Status* parameter to **adStatusUnwantedEvent** for **adRsnAddNew**, **adRsnDelete**, **adRsnUpdate**, **adRsnUndoUpdate**, **adRsnUndoAddNew**,

adRsnUndoDelete, and **adRsnFirstChange** as they occur.

Value	Description
adStatusUnwantedEvent	Request that this event handler receive no further notifications.
adStatusCancel	Request cancellation of the operation that is about to occur.

Error Parameter

The *Error* parameter is a reference to an ADO [Error](#) object. When the *Status* parameter is set to **adStatusErrorsOccurred**, the **Error** object contains details about why the operation failed. If the Will event associated with a Complete event has canceled the operation by setting the *Status* parameter to **adStatusCancel**, the error object is always set to **adErrOperationCancelled**.

Object Parameter

Each event receives one or more objects representing the objects that are involved in the operation. For example, the **ExecuteComplete** event receives a **Command** object, a **Recordset** object, and a **Connection** object.

Reason Parameter

The *Reason* parameter, *adReason*, provides additional information about why the event occurred. Events with an *adReason* parameter may be called several times, even for the same operation, each time for a different reason. For example, the **WillChangeRecord** event handler is called for operations that are about to do or undo the insertion, deletion, or modification of a record. If you want to process an event only when it occurs for a particular reason, you can use the *adReason* parameter to filter out the occurrences you are not interested in. For example, if you wanted to process record-change events only when they occur because a record was added, you can do something like this:

```
' BeginEventExampleVB01
Private Sub rsTest_WillChangeRecord(ByVal adReason As ADODB.EventRea
    If adReason = adRsnAddNew Then
        ' Process event
        ' ...
    Else
        ' Cancel event notification for all
        ' other possible adReason values.
        adStatus = adStatusUnwantedEvent
    End If
End Sub
' EndEventExampleVB01
```

In this case, notification can potentially occur for each of the other reasons. However, it will occur only once for each reason. After the notification has occurred once for each reason, you will receive notification only for the addition of a new record.

In contrast, you need to set *adStatus* to **adStatusUnwantedEvent** only one time to request that an event handler without an **adReason** parameter stop receiving event notifications.

See Also

[ADO Event Handler Summary](#) | [ADO Event Instantiation by Language](#) | [How Event Handlers Work Together](#) | [Types of Events](#)

ADO 2.5 

How Event Handlers Work Together

Unless you are programming in Visual Basic, all event handlers for **Connection** and **Recordset** events must be implemented, regardless of whether you actually process all of the events. The amount of implementation work you have to do depends on your programming language. For more information, see [ADO Event Instantiation by Language](#).

Paired Event Handlers

Each Will event handler has an associated Complete event handler. For example, when your application changes the value of a field, the **WillChangeField** event handler is called. If the change is acceptable, your application leaves the **adStatus** parameter unchanged and the operation is performed. When the operation completes, a **FieldChangeComplete** event notifies your application that the operation has finished. If it completed successfully, **adStatus** contains **adStatusOK**; otherwise, **adStatus** contains **adStatusErrorsOccurred** and you must check the **Error** object to determine the cause of the error.

When **WillChangeField** is called, you might determine that the change should not be made. In that case, set **adStatus** to **adStatusCancel**. The operation is canceled and the **FieldChangeComplete** event receives an **adStatus** value of **adStatusErrorsOccurred**. The **Error** object contains **adErrOperationCancelled** so that your **FieldChangeComplete** handler knows that the operation was canceled. However, you need to check the value of the **adStatus** parameter before changing it, because setting **adStatus** to **adStatusCancel** has no effect if the parameter was set to **adStatusCantDeny** on entry to the procedure.

Sometimes an operation can raise more than one event. For example, the **Recordset** object has paired events for **Field** changes and **Record** changes. When your application changes the value of a **Field**, the **WillChangeField** event handler is called. If it determines that the operation can continue, the **WillChangeRecord** event handler is also raised. If this handler also allows the event to continue, the change is made and the **FieldChangeComplete** and **RecordChangeComplete** event handlers are called. The order in which the Will event handlers for a particular operation are called is not defined, so you should avoid writing code that depends on calling handlers in a particular sequence.

In instances when multiple Will events are raised, one of the events might cancel the pending operation. For example, when your application changes the value of a **Field**, both **WillChangeField** and **WillChangeRecord** event handlers would normally be called. However, if the operation is canceled in the first event handler, its associated Complete handler is immediately called with **adStatusOperationCancelled**. The second handler is never called. If, however, the first event handler allows the event to proceed, the other event handler will

be called. If it then cancels the operation, both Complete events will be called as in the earlier examples.

Unpaired Event Handlers

As long as the status passed to the event is not **adStatusCantDeny**, you can turn off event notifications for any event by returning **adStatusUnwantedEvent** in the *Status* parameter. For example, when your Complete event handler is called the first time, you can return **adStatusUnwantedEvent**. You will subsequently receive only Will events. However, some events can be triggered for more than one reason. In that case, the event will have a *Reason* parameter. When you return **adStatusUnwantedEvent**, you will stop receiving notifications for that event only when they occur for that particular reason. In other words, you will potentially receive notification for each possible reason that the event could be triggered.

Single Will event handlers can be useful when you want to examine the parameters that will be used in an operation. You can modify those operation parameters or cancel the operation.

Alternatively, leave Complete event notification enabled. When your first Will event handler is called, return **adStatusUnwantedEvent**. You will subsequently receive only Complete events.

Single Complete event handlers can be useful for managing [asynchronous operations](#). Each asynchronous operation has an appropriate Complete event.

For example, it can take a long time to populate a large [Recordset](#) object. If your application is appropriately written, you can start a `Recordset.Open(..., adAsyncExecute)` operation and continue with other processing. You will eventually be notified when the **Recordset** is populated by an **ExecuteComplete** event.

Single Event Handlers and Multiple Objects

The flexibility of a programming language like Microsoft Visual C++ enables you to have one event handler process events from multiple objects. For example, you could have one **Disconnect** event handler process events from several **Connection** objects. If one of the connections ended, the **Disconnect** event handler would be called. You could tell which connection caused the event because the event-handler object parameter would be set to the corresponding **Connection** object.

Note This technique cannot be used in Visual Basic because that language can correlate only one object to an event handler.

See Also

[ADO Event Handler Summary](#) | [ADO Event Instantiation by Language](#) | [Event Parameters](#) | [Types of Events](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

ADO Event Instantiation by Language

Each programming language creates instances of ADO events differently. All of the following examples create a **ConnectComplete** event handler.

- [Visual Basic](#)
- [Visual C++](#)
- [Visual J++](#)
- [VBScript](#)
- [JScript](#)
- [ADO/WFC](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 


```

        ' Will event. Notify the user and exit.
        strMsg = "I'm sorry you can't connect right now.
        strMsg = strMsg & " Click OK to exit."
        Unload Me
    Case Else
        strMsg = "Error " & Format(pError.Number) & vbCr
        strMsg = strMsg & pError.Description
        strMsg = strMsg & " Click OK to exit."
        Unload Me
    End Select
Else
    strMsg = "Error occured. Click OK to exit."
    Unload Me
End If
End If
'frmWait.btnOK.Enabled = True
End Sub
' EndEventExampleVB02

```

The **Connection** object is declared at the **Form** level using the **WithEvents** keyword to enable event handling. The `Form_Load` event handler actually creates the object by assigning a new **Connection** object to `connEvent` and then opens the connection. Of course, a real application would do more processing in the `Form_Load` event handler than is shown here.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Visual C++

This is a schematic description of how to instantiate ADO events in Microsoft Visual C++. See [ADO Events Model Example \(VC++\)](#) for a complete description.

Create classes derived from the **ConnectionEventsVt** and **RecordsetEventsVt** interfaces found in the file `adoint.h`.

```
// BeginEventExampleVC01
class CConnEvent : public ConnectionEventsVt
{
    public:
        STDMETHODCALLTYPE InfoMessage(
            ADOError *pError,
            EventStatusEnum *adStatus,
            _ADOConnection *pConnection);
    ...
}

class CRstEvent : public RecordsetEventsVt
{
    public:
        STDMETHODCALLTYPE WillChangeField(
            LONG cFields,
            VARIANT Fields,
            EventStatusEnum *adStatus,
            _ADORecordset *pRecordset);
    ...
}
// EndEventExampleVC01
```

Implement each of the event-handler methods in both classes. It is sufficient that each method merely return an HRESULT of S_OK. However, when you make it known that your event handlers are available, they will be called continuously by default. Instead, you might want to request no further notification after the first time by setting **adStatus** to **adStatusUnwantedEvent**.

```
// BeginEventExampleVC02
STDMETHODIMP CConnEvent::ConnectComplete(
    ADOError *pError,
    EventStatusEnum *adStatus,
    _ADOConnection *pConnection)
```

```

    {
        *adStatus = adStatusUnwantedEvent;
        return S_OK;
    }

```

```
// EndEventExampleVC02
```

The event classes inherit from **IUnknown**, so you must also implement the **QueryInterface**, **AddRef**, and **Release** methods. Also implement class constructors and destructors. Choose the Visual C++ tools with which you are most comfortable to simplify this part of the task.

Make it known that your event handlers are available by issuing **QueryInterface** on the [Recordset](#) and [Connection](#) objects for the **IConnectionPointContainer** and **IConnectionPoint** interfaces. Then issue **IConnectionPoint::Advise** for each class.

For example, assume you are using a Boolean function that returns **True** if it successfully informs a **Recordset** object that you have event handlers available.

```

// BeginEventExampleVC03
HRESULT      hr;
DWORD       dwEvtClass;
IConnectionPointContainer    *pCPC = NULL;
IConnectionPoint             *pCP = NULL;
CRstEvent                    *pRStEvent = NULL;
...
_RecordsetPtr                pRs;
pRs.CreateInstance(__uuidof(Recordset));
pRStEvent = new CRstEvent;
if (pRStEvent == NULL) return FALSE;
...
hr = pRs->QueryInterface(IID_IConnectionPointContainer, (LPVOID *)&pCPC);
if (FAILED(hr)) return FALSE;
hr = pCPC->FindConnectionPoint(RecordsetEvents, &pCP);
pCPC->Release();    // Always Release now, even before checking.
if (FAILED(hr)) return FALSE;
hr = pCP->Advise(pRStEvent, &dwEvtClass);    //Turn on event support.
pCP->Release();
if (FAILED(hr)) return FALSE;
...
return TRUE;
...
// EndEventExampleVC03

```

At this point, events for the **RecordsetEvent** family are enabled and your

methods will be called as **Recordset** events occur.

Later, when you want to make your event handlers unavailable, get the connection point again and issue the **ICorrelationPoint::Unadvise** method.

```
// BeginEventExampleVC04
...
hr = pCP->Unadvise(dwEvtClass);    //Turn off event support.
pCP->Release();
if (FAILED(hr)) return FALSE;
...
// EndEventExampleVC04
```

You must release interfaces and destroy class objects as appropriate.

The following code shows a complete example of a **Recordset** Event sink class.

```
// BeginEventExampleVC05
#include <adoint.h>

class CADORecordsetEvents : public RecordsetEventsVt
{
public :

    ULONG m_ulRefCount;
    CADORecordsetEvents():m_ulRefCount(1){}

    STDMETHODCALLTYPE(QueryInterface)(REFIID iid, LPVOID * ppvObject)
    {
        if (IsEqualIID(__uuidof(IUnknown), iid) ||
            IsEqualIID(__uuidof(RecordsetEventsVt), iid))
        {
            *ppvObject = this;
            return S_OK;
        }
        else
            return E_NOINTERFACE;
    }

    STDMETHODCALLTYPE_(ULONG, AddRef)()
    {
        return m_ulRefCount++;
    }

    STDMETHODCALLTYPE_(ULONG, Release)()
    {
        if (--m_ulRefCount == 0)
        {
```

```

        delete this;
        return 0;
    }
    else
        return m_ulRefCount;
}

STDMETHOD(WillChangeField)(
    LONG cFields,
    VARIANT Fields,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}

STDMETHOD(FieldChangeComplete)(
    LONG cFields,
    VARIANT Fields,
    ADOError *pError,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}

STDMETHOD(WillChangeRecord)(
    EventReasonEnum adReason,
    LONG cRecords,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}

STDMETHOD(RecordChangeComplete)(
    EventReasonEnum adReason,
    LONG cRecords,
    ADOError *pError,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}

```

```
STDMETHOD(WillChangeRecordset)(
    EventReasonEnum adReason,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}
```

```
STDMETHOD(RecordsetChangeComplete)(
    EventReasonEnum adReason,
    ADOError *pError,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}
```

```
STDMETHOD(WillMove)(
    EventReasonEnum adReason,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}
```

```
STDMETHOD(MoveComplete)(
    EventReasonEnum adReason,
    ADOError *pError,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}
```

```
STDMETHOD(EndOfRecordset)(
    VARIANT_BOOL *fMoreData,
    EventStatusEnum *adStatus,
    _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}
```

```
    STDMETHODCALLTYPE(
        long Progress,
        long MaxProgress,
        EventStatusEnum *adStatus,
        _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}

    STDMETHODCALLTYPE(
        ADOError *pError,
        EventStatusEnum *adStatus,
        _ADORecordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
}
};
// EndEventExampleVC05
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Visual J++

This short Microsoft Visual J++ example shows how you can associate your own function with a particular event.

```
// BeginEventExampleVJ
import com.ms.wfc.data.*;

public class EventExampleVJ
{
    ConnectionEventHandler handler = new ConnectionEventHandler(this

    public void onConnectComplete(Object sender, ConnectionEvent e)
    {
        if (e.adStatus == AdoEnums.EventStatus.ERRORSOCCURRED)
            System.out.println("Connection failed");
        else
            System.out.println("Connection completed");
        return;
    }

    public static void main (String[] args)
    {
        EventExampleVJ Class1 = new EventExampleVJ();
        Connection conn = new Connection();

        conn.addOnConnectComplete(Class1.handler);    // Enable eve
        conn.open("DSN=Pubs");
        conn.close();
        conn.removeOnConnectComplete(Class1.handler); // Disable ev
    }
}
// EndEventExampleVJ
```

First, the class method *onConnectionComplete* is associated with the **ConnectionComplete** event by creating a new **ConnectionEventHandler** object and assigning the *onConnectComplete* function to the object.

The *main* function then creates a **Connection** object and enables event handling by calling the **addOnConnectComplete** method and passing it the address of the *handler* function.

ADO 2.5 

VBScript

Microsoft Visual Basic Scripting Edition does not support ADO events.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

JScript

Microsoft JScript does not support ADO events.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

ADO/WFC

ADO for Windows Foundation Classes (ADO/WFC) builds on the ADO event model and presents a simplified application programming interface. In general, ADO/WFC intercepts ADO events, consolidates the event parameters into a single event class, and then calls your event handler.

To use ADO events in ADO/WFC

1. Define your own event handler to process an event. For example, if you wanted to process the **ConnectComplete** event in the **ConnectionEvent** family, you might use this code:

```
public void onConnectComplete(Object sender, ConnectionEvent e)
{
    System.out.println("onConnectComplete:" + e);
}
```

2. Define a handler object to represent your event handler. The handler object should be of data type **ConnectEventHandler** for an event of type **ConnectionEvent**, or data type **RecordsetEventHandler** for an event of type **RecordsetEvent**. For example, code the following for your **ConnectComplete** event handler:

```
ConnectionEventHandler handler =
    new ConnectionEventHandler(this, "onConnectComplete");
```

The first argument of the **ConnectionEventHandler** constructor is a reference to the class that contains the event handler named in the second argument.

The Microsoft Visual J++ compiler also supports an equivalent syntax:

```
ConnectionEventHandler handler =
    new ConnectionEventHandler(this.onConnectComplete);
```

The single argument is a reference to the desired class (**this**) and method within the class (**onConnectComplete**).

3. Add your event handler to a list of handlers designated to process a

particular type of event. Use the method with a name such as **addOnEventName(handler)**.

4. ADO/WFC internally implements all the ADO event handlers. Therefore, an event caused by a **Connection** or **Recordset** operation is intercepted by an ADO/WFC event handler.

The ADO/WFC event handler passes ADO **ConnectionEvent** parameters in an instance of the ADO/WFC **ConnectionEvent** class, or ADO **RecordsetEvent** parameters in an instance of the ADO/WFC **RecordsetEvent** class. These ADO/WFC classes consolidate the ADO event parameters; that is, each ADO/WFC class contains one data member for each unique parameter in all the ADO **ConnectionEvent** or **RecordsetEvent** methods.

5. ADO/WFC then calls your event handler with the ADO/WFC event object. For example, your **onConnectComplete** handler has a signature like this:

```
public void onConnectComplete(Object sender, ConnectionEvent
```

The first argument is the type of object that sent the event ([Connection](#) or [Recordset](#)), and the second argument is the ADO/WFC event object (**ConnectionEvent** or **RecordsetEvent**).

The signature of your event handler is simpler than an ADO event. However, you must still understand the ADO event model to know what parameters apply to an event and how to respond.

6. Return from your event handler to the ADO/WFC handler for the ADO event. ADO/WFC copies pertinent ADO/WFC event data members back to the ADO event parameters, and then the ADO event handler returns.
7. When you are finished processing, remove your handler from the list of ADO/WFC event handlers. Use the method with a name such as **removeOnEventName(handler)**.

See Also

[ADO Event Handler Summary](#) | [ADO/WFC Programming](#) | [ADO/WFC Syntax Index](#) | [Event Parameters](#) | [How Event Handlers Work Together](#) | [Types of Events](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

Chapter 8: Understanding Cursors and Locks

It is important to understand how cursors operate so you can select the best and most efficient cursor type for an application's data-access requirements. A less-than-optimal cursor configuration can make data-access operations painfully slow.

Many capabilities of the ADO **Recordset** object are determined by the type and location of the cursor, as well as the lock type.

This chapter covers the following topics:

- [What is a Cursor?](#)
- [Types of Cursors](#)
- [The Significance of Cursor Location](#)
- [The Microsoft Cursor Service for OLE DB](#)
- [What is a Lock?](#)
- [Using CacheSize](#)
- [Cursor and Lock Characteristics](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

What is a Cursor?

Operations in a relational database act on a complete set of rows. 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. This complete set of rows returned by the statement is known as the result set. Applications, especially those that are interactive and online, cannot always work effectively with the entire result set as a unit. These applications need a mechanism to work with one row or a small block of rows at a time. Cursors are an extension to result sets that provide that mechanism.

A cursor is implemented by a cursor library. A cursor library is software, often implemented as a part of a database system or a data access API, that is used to manage attributes of data returned from a data source (a result set). These attributes include concurrency management, position in the result set, number of rows returned, and whether or not you can move forward and/or backward through the result set (scrollability).

A cursor keeps track of the position in the result set, and allows you to perform multiple operations row by row against a result set, with or without returning to the original table. In other words, cursors conceptually return a result set based on tables within the databases. The cursor is so named because it indicates the current position in the result set, just as the cursor on a computer screen indicates current position.

It is important to become familiar with the concept of cursors before moving on to learn the specifics of their usage in ADO.

Using cursors, you can:

- Specify positioning at specific rows in the result set.
- Retrieve one row or a block of rows based on the current result set position.
- Modify data in the rows at the current position in the result set.
- Define different levels of sensitivity to data changes made by other users.

For example, consider an application that displays a list of available products to a potential buyer. The buyer scrolls through the list to see product details and

cost, and finally selects a product for purchase. Additional scrolling and selection occurs for the remainder of the list. As far as the buyer is concerned, the products appear one at a time, but the application uses a scrollable cursor to browse up and down through the result set.

You can use cursors in a variety of ways:

- With no rows at all.
- With some or all of the rows in a single table.
- With some or all of the rows from logically joined tables.
- As read-only or updateable at the cursor or field level.
- As forward-only or fully scrollable.
- With the cursor keyset located on the server.
- Sensitive to underlying table changes caused by other applications (such as membership, sort, inserts, updates, and deletes).
- Existing on either the server or the client.

Read-only cursors help users browse through the result set, and read/write cursors can implement individual row updates. Complex cursors can be defined with keysets that point back to base table rows. While some cursors are read-only in a forward direction, others can move back and forth and provide a dynamic refresh of the result set based on changes that other applications are making to the database.

Not all applications need to use cursors to access or update data. Some queries simply do not require direct row updating by using a cursor. Cursors should be one of the last techniques you choose to retrieve data—and then you should choose the lowest impact cursor possible. When you create a result set by using a stored procedure, the result set is not updateable using cursor edit or update methods.

Concurrency

In some multi-user applications it is vitally important for the data presented to the end user to be as current as possible. A classic example of such a system is an airline reservation system, where many users might be contending for the same seat on a given flight (and thus, a single record). In a case like this, the application design must handle concurrent operations on a single record.

In other applications, concurrency is not as important. In such cases, the expense involved in keeping the data current at all times cannot be justified.

Position

A cursor also keeps track of the current position in a result set. Think of the cursor position as a pointer to the current record, similar to the way an array index points to the value at that particular location in the array.

Scrollability

The type of cursor employed by your application also affects the ability to move forward and backward through the rows in a result set; this is sometimes called scrollability. The ability to move forward *and* backward through a result set adds to the complexity of the cursor, and is therefore more expensive to implement. For this reason, you should ask for a cursor with this functionality only when necessary.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Types of Cursors

As a general rule, your application should use the simplest cursor that provides the required data access. Each additional cursor characteristic beyond the basics (forward-only, read-only, static, scrolling, unbuffered) has a price—in client memory, network load, or performance. In many cases, the default cursor options generate a more complex cursor than your application actually needs.

Your choice of cursor type depends on how your application uses the result set and also on several design considerations, including the size of the result set, the percentage of the data likely to be used, sensitivity to data changes, and application performance requirements.

At its most basic, your cursor choice depends on whether you need to change or simply view the data:

- If you just need to scroll through a set of results, but not change data, use a [forward-only](#) or [static](#) cursor.
- If you have a large result set and need to select just a few rows, use a [keyset](#) cursor.
- If you want to synchronize a result set with recent adds, changes, and deletes by all concurrent users, use a [dynamic](#) cursor.

Although each cursor type seems to be distinct, keep in mind that these cursor types are not so much different varieties as simply the result of overlapping characteristics and options.

See Also

[Forward-Only Cursors](#) | [Static Cursors](#) | [Keyset Cursors](#) | [Dynamic Cursors](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Forward-Only Cursors

The typical default cursor type, called a forward-only (or non-scrollable) cursor, can move only forward through the result set. A forward-only cursor does not support scrolling (the ability to move forward and backward in the result set); it only supports fetching rows from the start to the end of the result set. With some forward-only cursors (such as with the SQL Server cursor library), all insert, update, and delete statements made by the current user (or committed by other users) that affect rows in the result set are visible as the rows are fetched.

Because the cursor cannot be scrolled backward, however, changes made to rows in the database after the row was fetched are not visible through the cursor.

After the data for the current row is processed, the forward-only cursor releases the resources that were used to hold that data. Forward-only cursors are dynamic by default, meaning that all changes are detected as the current row is processed. This provides faster cursor opening and enables the result set to display updates made to the underlying tables.

While forward-only cursors do not support backward scrolling, your application can return to the beginning of the result set by closing and reopening the cursor. This is an effective way to work with small amounts of data. As an alternative, your application could read the result set once, cache the data locally, and then browse the local data cache.

If your application does not require scrolling through the result set, the forward-only cursor is the best way to retrieve data quickly with the least amount of overhead. Use the **adOpenForwardOnly CursorTypeEnum** to indicate that you want to use a forward-only cursor in ADO.

See Also

[Static Cursors](#) | [Keyset Cursors](#) | [Dynamic Cursors](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Static Cursors

The static cursor always displays the result set as it was when the cursor was first opened. Depending on implementation, static cursors are either read-only or read/write and provide forward and backward scrolling. The static cursor does not usually detect changes made to the membership, order, or values of the result set after the cursor is opened. Static cursors may detect their own updates, deletes, and inserts, although they are not required to do so.

Static cursors never detect other updates, deletes, and inserts. For example, suppose a static cursor fetches a row, and another application then updates that row. If the application refetches the row from the static cursor, the values it sees are unchanged, despite the changes made by the other application. All types of scrolling are supported, but providers may or may not support bookmarks.

If your application does not need to detect data changes and requires scrolling, the static cursor is the best choice. Use the **adOpenStatic CursorTypeEnum** to indicate that you want to use a static cursor in ADO.

See Also

[Forward-Only Cursors](#) | [Keyset Cursors](#) | [Dynamic Cursors](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Keyset Cursors

The keyset cursor provides functionality between a static and a dynamic cursor in its ability to detect changes. Like a static cursor, it does not always detect changes to the membership and order of the result set. Like a dynamic cursor, it does detect changes to the values of rows in the result set.

Keyset-driven cursors are controlled by a set of unique identifiers (keys) known as the keyset. The keys are built from a set of columns that uniquely identify the rows in the result set. The keyset is the set of key values from all the rows returned by the query statement.

With keyset-driven cursors, a key is built and saved for each row in the cursor and stored either on the client workstation or on the server. When you access each row, the stored key is used to fetch the current data values from the data source. In a keyset-driven cursor, result set membership is frozen when the keyset is fully populated. Thereafter, additions or updates that affect membership are not a part of the result set until it is reopened.

Changes to data values (made either by the keyset owner or other processes) are visible as the user scrolls through the result set. Inserts made outside the cursor (by other processes) are visible only if the cursor is closed and reopened. Inserts made from inside the cursor are visible at the end of the result set.

When a keyset-driven cursor attempts to retrieve a row that has been deleted, the row appears as a "hole" in the result set. The key for the row exists in the keyset, but the row no longer exists in the result set. If the key values in a row are updated, the row is considered to have been deleted and then inserted, so such rows also appear as holes in the result set. While a keyset-driven cursor can always detect rows deleted by other processes, it can optionally remove the keys for rows it deletes itself. Keyset-driven cursors that do this cannot detect their own deletes because the evidence has been removed.

An update to a key column operates like a delete of the old key followed by an insert of the new key. The new key value is not visible if the update was not made through the cursor. If the update was made through the cursor, the new key value is visible at the end of the result set.

There is a variation on keyset-driven cursors called keyset-driven standard cursors. In a keyset-driven standard cursor, the membership of rows in the result set and the order of the rows are fixed at cursor open time, but changes to values that are made by the cursor owner and committed changes made by other processes are visible. If a change disqualifies a row for membership or affects the order of a row, the row does not disappear or move unless the cursor is closed and reopened. Inserted data does not appear, but changes to existing data do appear as the rows are fetched.

The keyset-driven cursor is difficult to use correctly because the sensitivity to data changes depends on many differing circumstances, as described above. However, if your application is not concerned with concurrent updates, can programmatically handle bad keys, and must directly access certain keyed rows, the keyset-driven cursor might work for you. Use the **adOpenKeyset CursorTypeEnum** to indicate that you want to use a keyset cursor in ADO.

See Also

[Forward-Only Cursors](#) | [Static Cursors](#) | [Dynamic Cursors](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Dynamic Cursors

Dynamic cursors detect all changes made to the rows in the result set, regardless of whether the changes occur from inside the cursor or by other users outside the cursor. All insert, update, and delete statements made by all users are visible through the cursor. The dynamic cursor can detect any changes made to the rows, order, and values in the result set after the cursor is opened. Updates made outside the cursor are not visible until they are committed (unless the cursor transaction isolation level is set to "uncommitted").

For example, suppose a dynamic cursor fetches two rows and another application, and then updates one of those rows and deletes the other. If the dynamic cursor then fetches those rows, it will not find the deleted row, but it will display the new values for the updated row.

The dynamic cursor is a good choice if your application must detect all concurrent updates made by other users. Use the **adOpenDynamic CursorTypeEnum** to indicate that you want to use a dynamic cursor in ADO.

[Forward-Only Cursors](#) | [Static Cursors](#) | [Keyset Cursors](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

The Significance of Cursor Location

Every cursor uses temporary resources to hold its data. These resources can be memory, a disk paging file, temporary disk files, or even temporary storage in the database. The cursor is called a *client-side* cursor when these resources are located on the client computer. The cursor is called a *server-side* cursor when these resources are located on the server.

Client-Side Cursors

In ADO, call for a client-side cursor by using the **adUseClient CursorLocationEnum**. With a non-keyset client-side cursor, the server sends the entire result set across the network to the client computer. The client computer provides and manages the temporary resources needed by the cursor and result set. The client-side application can browse through the entire result set to determine which rows it requires.

Static and keyset-driven client-side cursors may place a significant load on your workstation if they include too many rows. While all of the cursor libraries are capable of building cursors with thousands of rows, applications designed to fetch such large rowsets may perform poorly. There are exceptions, of course. For some applications, a large client-side cursor might be perfectly appropriate and performance might not be an issue.

One obvious benefit of the client-side cursor is quick response. After the result set has been downloaded to the client computer, browsing through the rows is very fast. Your application is generally more scalable with client-side cursors because the cursor's resource requirements are placed on each separate client and not on the server.

Server-Side Cursors

In ADO, call for a server-side cursor by using the **adUseServerCursorLocationEnum**. With a server-side cursor, the server manages the result set using resources provided by the server computer. The server-side cursor returns only the requested data over the network. This type of cursor can sometimes provide better performance than the client-side cursor, especially in situations where excessive network traffic is a problem.

However, it is important to point out that a server-side cursor is—at least temporarily—consuming precious server resources for every active client. You must plan accordingly to ensure that your server hardware is capable of managing all of the server-side cursors requested by active clients. Also, a server-side cursor can be slow because it provides only single row access—there is no batch cursor available.

Server-side cursors are useful when inserting, updating, or deleting records. With server-side cursors, you can have multiple active statements on the same connection.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

The Microsoft Cursor Service for OLE DB

When you select a client-side cursor, or set the **CursorLocation** property to **adUseClient**, you are invoking the Microsoft Cursor Service for OLE DB. You might also see references to the "Client Cursor Engine", which is essentially the same thing in the context of ADO. This service supplements the cursor-support functions of data providers. As a result, you can perceive relatively uniform functionality from all data providers.

The Cursor Service for OLE DB makes dynamic properties available and enhances the behavior of certain methods. For example, the **Optimize** dynamic property enables the creation of temporary indexes to facilitate certain operations, such as the **Find** method.

The Cursor Service enables support for batch updating in all cases. It also simulates more capable cursor types, such as dynamic cursors, when a data provider can only supply less capable cursors, such as static cursors.

See Also

[Microsoft Cursor Service for OLE DB](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

What is a Lock?

Locking is the process by which a DBMS restricts access to a row in a multi-user environment. When a row or column is exclusively locked, other users are not permitted to access the locked data until the lock is released. This ensures that two users cannot simultaneously update the same column in a row.

Locks can be very expensive from a resource perspective and should be used only when required to preserve data integrity. In a database where hundreds or thousands of users could be trying to access a record every second—such as a database connected to the Internet—unnecessary locking could quickly result in slower performance in your application.

You can control how the data source and the ADO cursor library manage concurrency by choosing the appropriate locking option.

Set the **LockType** property before opening a **Recordset** to specify what type of locking the provider should use when opening it. Read the property to return the type of locking in use on an open **Recordset** object.

Providers might not support all lock types. If a provider cannot support the requested **LockType** setting, it will substitute another type of locking. To determine the actual locking functionality available in a **Recordset** object, use the [Supports](#) method with **adUpdate** and **adUpdateBatch**.

The **adLockPessimistic** setting is not supported if the [CursorLocation](#) property is set to **adUseClient**. If an unsupported value is set, no error will result; the closest supported **LockType** will be used instead.

The **LockType** property is read/write when the **Recordset** is closed, and read-only when it is open.

ADO 2.5 

Types of Locks

adLockBatchOptimistic

Indicates optimistic batch updates. Required for batch update mode.

Many applications fetch a number of rows at once and then need to make coordinated updates that include the entire set of rows to be inserted, updated, or deleted. With batch cursors, only one round trip to the server is needed, thus improving update performance and decreasing network traffic. Using a batch cursor library, you can create a static cursor and then disconnect from the data source. At this point you can make changes to the rows and subsequently reconnect and post the changes to the data source in a batch.

adLockOptimistic

Indicates that the provider uses optimistic locking—locking records only when you call the **Update** method. This means that there is a chance that the data may be changed by another user between the time you edit the record and when you call **Update**, which creates conflicts. Use this lock type in situations where the chances of a collision are low or where collisions can be readily resolved.

adLockPessimistic

Indicates pessimistic locking, record by record. The provider does what is necessary to ensure successful editing of the records, usually by locking records at the data source immediately before editing. Of course, this means that the records are unavailable to other users once you begin to edit, until you release the lock by calling **Update**. Use this type of lock in a system where you cannot afford to have concurrent changes to data, such as in a reservation system.

adLockReadOnly

Indicates read-only records. You cannot alter the data. A read-only lock is the "fastest" type of lock because it does not require the server to maintain a lock on the records.

adLockUnspecified

Does not specify a type of lock.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Using CacheSize

Use the **CacheSize** property to control how many records to retrieve at one time into local memory from the provider. For example, if the **CacheSize** is 10, after first opening the **Recordset** object, the provider retrieves the first 10 records into local memory. As you move through the **Recordset** object, the provider returns the data from the local memory buffer. As soon as you move past the last record in the cache, the provider retrieves the next 10 records from the data source into the cache.

Note **CacheSize** is based on the **Maximum Open Rows** provider-specific property (in the **Properties** collection of the **Recordset** object). You cannot set **CacheSize** to a value greater than **Maximum Open Rows**. To modify the number of rows that can be opened by the provider, set **Maximum Open Rows**.

The value of **CacheSize** can be adjusted during the life of the **Recordset** object, but changing this value only affects the number of records in the cache after subsequent retrievals from the data source. Changing the property value alone will not change the current contents of the cache.

If there are fewer records to retrieve than **CacheSize** specifies, the provider returns the remaining records and no error occurs.

A **CacheSize** setting of zero is not allowed and returns an error.

Records retrieved from the cache do not reflect concurrent changes that other users made to the source data. To force an update of all the cached data, use the [Resync](#) method.

If **CacheSize** is set to a value greater than 1, the navigation methods ([Move](#), [MoveFirst](#), [MoveLast](#), [MoveNext](#), and [MovePrevious](#)) may result in navigation to a deleted record, if deletion occurs after the records were retrieved. After the initial fetch, subsequent deletions will not be reflected in your data cache until you attempt to access a data value from a deleted row. However, setting **CacheSize** to 1 eliminates this issue because deleted rows cannot be fetched.

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

Cursor and Lock Characteristics

While the characteristics of a cursor depend upon capabilities of the provider, the following advantages and disadvantages generally apply to the various types of cursors and locks.

Cursor or lock type	Advantages	Disadvantages
adOpenForwardOnly	<ul style="list-style-type: none">• Low resource requirements	<ul style="list-style-type: none">• Cannot scroll backward• No data concurrency
adOpenStatic	<ul style="list-style-type: none">• Scrollable	<ul style="list-style-type: none">• No data concurrency
adOpenKeyset	<ul style="list-style-type: none">• Some data concurrency• Scrollable	<ul style="list-style-type: none">• Higher resource requirements• Not available in disconnected scenario
adOpenDynamic	<ul style="list-style-type: none">• High data concurrency• Scrollable	<ul style="list-style-type: none">• Highest resource requirements• Not available in disconnected scenario
adLockReadOnly	<ul style="list-style-type: none">• Low resource requirements• Highly scalable	<ul style="list-style-type: none">• Data not updatable through cursor
adLockBatchOptimistic	<ul style="list-style-type: none">• Batch updates• Allows disconnected scenarios• Other users able to	<ul style="list-style-type: none">• Data can be changed by multiple users at once

access data

adLockPessimistic

- Data cannot be changed by other users while locked
- Prevents other users from accessing data while locked

adLockOptimistic

- Other users able to access data
- Data can be changed by multiple users at once

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Chapter 9: Data Shaping

Data shaping provides a way to query a data source and return a [Recordset](#) that represents a parent-child relationship between two or more logical entities (a hierarchy). A classic example of a hierarchical relationship is customers and orders. For every customer in a database, there can be zero or more orders. Regular SQL provides a means of retrieving the data using JOIN syntax, but this can be inefficient and unwieldy because redundant parent data is repeated in each record returned for a given parent-child relationship. Data shaping can relate a single parent record in the parent **Recordset** to multiple child records in the child **Recordset**, avoiding the redundancy of a JOIN. Most people find the parent-child multiple **Recordset** programming model more natural and easier to work with than the single **Recordset** JOIN model.

The data shaping syntax also provides other capabilities. Developers can create new **Recordset** objects without an underlying data source by using the NEW keyword to describe the fields of the parent and child **Recordsets**. The new **Recordset** object can be populated with data and persistently stored. Developers can also perform various calculations or aggregations (for example, SUM, AVG, and MAX) on child fields. Data shaping can also create a parent **Recordset** from a child **Recordset** by grouping records in the child and placing one row in the parent for each group in the child.

See the following topics to learn more about data shaping:

- [Data Shaping Summary](#)
- [Required Providers for Data Shaping](#)
- [Shape Commands in General](#)
- [Shape APPEND Clause](#)
- [Shape COMPUTE Clause](#)
- [Fabricating Hierarchical Recordsets](#)
- [Accessing Rows in a Hierarchical Recordset](#)
- [Formal Shape Grammar](#)
- [Visual Basic for Applications Functions](#)

ADO 2.5 

Data Shaping Summary

The following sections describe concepts of data shaping, hierarchical recordsets, reshaping, grandchild aggregates, parameterized shapes, and saving shapes to files.

- [Data Shaping](#)
- [Reshaping](#)
- [Grandchild Aggregates](#)
- [Parameterized Commands with Intervening COMPUTE Commands](#)
- [Persisting Hierarchical Recordsets](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Data Shaping

Data shaping enables you to define the columns of a shaped **Recordset**, the relationships between the entities represented by the columns, and the manner in which the **Recordset** is populated with data.

Columns of a shaped **Recordset** may contain data from a data provider such as Microsoft SQL Server, references to another **Recordset**, values derived from a calculation on a single row of a **Recordset**, values derived from an operation over a column of an entire **Recordset**; or they may be a newly fabricated, empty column.

When you retrieve the value of a column that contains a reference to another **Recordset**, ADO automatically returns the actual **Recordset** represented by the reference. A **Recordset** that contains another **Recordset** is called a hierarchical recordset. Hierarchical recordsets exhibit a *parent-child* relationship, in which the *parent* is the containing recordset and the *child* is the contained recordset. The reference to a **Recordset** is actually a reference to a subset of the child, called a *chapter*. A single parent may reference more than one child **Recordset**.

The shape command syntax enables you to programmatically create a shaped **Recordset**. You can then access the components of the **Recordset** programmatically or through an appropriate visual control. A shape command is issued like any other ADO command text.

You can make hierarchical **Recordset** objects in two ways with the shape command syntax. The first appends a child **Recordset** to a parent **Recordset**. The parent and child typically have at least one column in common: the value of the column in a row of the parent is the same as the value of the column in all rows of the child.

The second way generates a parent **Recordset** from a child **Recordset**. The records in the child **Recordset** are grouped, typically using the BY clause, and one row is added to the parent **Recordset** for each resulting group in the child. If the BY clause is omitted, the child **Recordset** will form a single group and the parent **Recordset** will contain exactly one row. This is useful for computing "grand total" aggregates over the entire child **Recordset**.

Regardless of which way the parent **Recordset** is formed, it will contain a chapter column that is used to relate it to a child **Recordset**. If you wish, the parent **Recordset** may also have columns that contain aggregates (SUM, MIN, MAX, and so on) over the child rows. Both the parent and the child **Recordset** may have columns which contain an expression on the row in the **Recordset**, as well as columns which are new and initially empty.

You can nest hierarchical **Recordset** objects to any depth (that is, create child **Recordset** objects of child **Recordset** objects, and so on).

You can access the **Recordset** components of the shaped **Recordset** programmatically or through an appropriate visual control.

Microsoft provides a visual tool that generates shape commands (see [The Data Environment Designer](#) in the Visual Basic documentation) and another that displays hierarchical cursors (see [Using the Microsoft Hierarchical Flexgrid Control](#) in the Visual Basic documentation).

For examples of shape commands and their resulting hierarchies, see Using the Data Shaping Service for OLE DB: A Closer Look.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Reshaping

A **Recordset** created by a clause of a shape command may be assigned an *alias* name (typically with the AS keyword). The alias of a shaped **Recordset** can be referenced in an entirely different command. That is, you may reuse, or *reshape*, a previously shaped **Recordset** in a new shape command. To support this feature, ADO provides a property, [Reshape Name](#).

Reshaping has two main functions. The first is to associate an existing **Recordset** with a new parent **Recordset**.

Example

```
. . .
rs1.Open "SHAPE {select * from Customers} " & _
        "APPEND ({select * from Orders} AS chapOrders " & _
        "RELATE CustomerID to CustomerID)", cn

rs2.Open "SHAPE {select * from Employees} " & _
        "APPEND (chapOrders RELATE EmployeeID to EmployeeID)", cn
. . .
```

The second function is to enable non-chaptered access to existing child **Recordset** objects, using the syntax "SHAPE <recordset reshape name>".

Note You cannot append columns to an existing **Recordset**, reshape a parameterized **Recordset** or the **Recordset** objects in any intervening COMPUTE clause, or perform aggregate operations on any **Recordset** descendant from the **Recordset** being reshaped. The **Recordset** being reshaped and the new shape command must both use the same [Connection](#).

See Also

[Data Shaping](#)

ADO 2.5 

Grandchild Aggregates

The chapter column created in a clause of a shape command may be given a *chapter-alias name* (typically with the AS keyword). You may identify any column in any chapter of the shaped **Recordset** with a fully qualified name identifying the child containing the column. For example, if the parent chapter, chap1, contains a child chapter, chap2, that has an amount column, amt, then the qualified name would be chap1.chap2.amt. The qualified name may then be used as an argument to one of the aggregate functions (SUM, AVG, MAX, MIN, COUNT, STDEV, or ANY).

See Also

[Data Shaping](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Parameterized Commands with Intervening COMPUTE Commands

A typical parameterized shape APPEND command has a clause that creates a parent **Recordset** with a query command and another clause that creates a child **Recordset** with a parameterized query command—that is, a command containing a parameter placeholder (a question mark, "?"). The resulting shaped **Recordset** has two levels, in which the parent occupies the upper level and the child occupies the lower level.

The clause that creates the child **Recordset** may now be an arbitrary number of nested shape COMPUTE commands, where the most deeply nested command contains the parameterized query. The resulting shaped **Recordset** has multiple levels, in which the parent occupies the uppermost level, the child occupies the lowermost level, and an arbitrary number of **Recordsets** generated by the shape COMPUTE commands occupy the intervening levels.

The typical use for this feature is to invoke the aggregate function and grouping abilities of shape COMPUTE commands to create intervening **Recordset** objects with analytical information about the child **Recordset**. Furthermore, because this is a parameterized shape command, each time a chapter column of the parent is accessed, a new child **Recordset** may be retrieved. Because the intervening levels are derived from the child, they also will be recomputed.

See Also

[Data Shaping](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Persisting Hierarchical Recordsets

You can save a hierarchical **Recordset** to a file in either ADTG or XML format by calling the [Save](#) method. However, two limitations apply when saving hierarchical **Recordsets** in XML format: You cannot save in XML if the hierarchical **Recordset** contains pending updates, and you cannot save a parameterized hierarchical **Recordset**.

For more information about the Data Shaping provider, see [Microsoft Data Shaping Service for OLE DB](#) (ADO) and The Data Shaping Service for OLE DB(OLE DB).

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Required Providers for Data Shaping

Data shaping typically requires two providers. The service provider, [Data Shaping Service for OLE DB](#), supplies the data shaping functionality, and a [data provider](#), such as the OLE DB Provider for SQL Server, supplies rows of data to populate the shaped [Recordset](#).

The name of the [service provider](#) (MSDataShape) can be specified as the value of the [Connection](#) object [Provider](#) property or the connection string keyword "Provider=MSDataShape;".

The name of the data provider can be specified as the value of the **Data Provider** dynamic property, which is added to the **Connection** object [Properties](#) collection by the Data Shaping Service for OLE DB, or the connection string keyword "**Data Provider=provider**".

No data provider is required if the **Recordset** is not populated (for example, as in a fabricated **Recordset** where columns are created with the NEW keyword). In that case, specify "**Data Provider=none**;".

Example

```
Dim cnn As New ADODB.Connection
cnn.Provider = "MSDataShape"
cnn.Open "Data Provider=SQLOLEDB;Integrated Security=SSPI;Database=N
```

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

Shape Commands in General

Data shaping defines the columns of a shaped **Recordset**, the relationships between the entities represented by the columns, and the manner in which the **Recordset** is populated with data.

A shaped **Recordset** may consist of the following types of columns.

Column type	Description
data	Fields from a Recordset returned by a query command to a data provider , table, or previously shaped Recordset .
chapter	A reference to another Recordset , called a <i>chapter</i> . Chapter columns make it possible to define a <i>parent-child</i> relationship where the <i>parent</i> is the Recordset containing the chapter column and the <i>child</i> is the Recordset represented by the chapter.
aggregate	The value of the column is derived by executing an <i>aggregate function</i> on all the rows or a column of all the rows of a child Recordset . (See Aggregate Functions in the following topic, Aggregate Functions, the CALC Function, and the NEW Keyword .)
calculated expression	The value of the column is derived by calculating a Visual Basic for Applications expression on columns in the same row of the Recordset . The expression is the argument to the CALC function. (See Calculated Expression in the following topic, Aggregate Functions, the CALC Function, and the NEW Keyword and in Visual Basic for Applications Functions .)
new	Empty, fabricated fields, which may be populated with data at a later time. The column is defined with the NEW keyword. (See NEW keyword in the following topic, Aggregate Functions, the CALC Function, and the NEW Keyword .)

A shape command may contain a clause specifying a query command to an

underlying data provider that will return a **Recordset** object. The query's syntax depends on the requirements of the underlying data provider. This will usually be Structured Query Language (SQL), although ADO does not require the use of any particular query language.

You could use a SQL JOIN clause to relate two tables; however, a hierarchical **Recordset** may represent the information more efficiently. Each row of a **Recordset** created by a JOIN repeats information redundantly from one of the tables. A hierarchical **Recordset** has only one [parent Recordset](#) for each of multiple [child Recordset](#) objects.

Shape commands can be issued by **Recordset** objects or by setting the [CommandText](#) property of the [Command](#) object and then calling the [Execute](#) method.

Shape commands can be nested. That is, the *parent-command* or *child-command* may itself be another shape command.

The shape provider always returns a client cursor, even when the user specifies a cursor location of **adUseServer**.

For information about navigating a hierarchical **Recordset**, see [Accessing Rows in a Hierarchical Recordset](#).

For precise information about syntactically correct shape commands, see [Formal Shape Grammar](#).

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Aggregate Functions, the CALC Function, and the NEW Keyword

Data shaping supports the following functions. The name assigned to the chapter containing the column to be operated on is the *chapter-alias*.

A chapter-alias may be fully qualified, consisting of each chapter column name leading to the chapter containing the *column-name*, all separated by periods. For example, if the parent chapter, chap1, contains a child chapter, chap2, that has an amount column, amt, then the qualified name would be chap1.chap2.amt.

Aggregate Functions	Description
SUM(<i>chapter-alias.column-name</i>)	Calculates the sum of all values in the specified column.
AVG(<i>chapter-alias.column-name</i>)	Calculates the average of all values in the specified column.
MAX(<i>chapter-alias.column-name</i>)	Calculates the maximum value in the specified column.
MIN(<i>chapter-alias.column-name</i>)	Calculates the minimum value in the specified column.
COUNT(<i>chapter-alias[.column-name]</i>)	Counts the number of rows in the specified alias. If a column is specified, only rows for which that column is non-Null are included in the count.
STDEV(<i>chapter-alias.column-name</i>)	Calculates the standard deviation in the specified column. A value of the specified column. ANY has a predictable value only when the value of the column is the same for all rows in the chapter.
ANY(<i>chapter-alias.column-name</i>)	Note If the column does not contain the same value for all of the rows in the chapter, the

SHAPE command arbitrarily returns one of the values to be the value of the ANY function.

Calculated expression	Description
CALC(<i>expression</i>)	Calculates an arbitrary expression, but only on the row of the Recordset containing the CALC function. Any expression using these Visual Basic for Applications (VBA) Functions is allowed.

NEW keyword	Description
NEW <i>field-type</i> [(<i>width</i> <i>scale</i> <i>precision</i> <i>error</i> [, <i>scale</i> <i>error</i>])]	Adds an empty column of the specified type to the Recordset .

The *field-type* passed with the NEW keyword can be any of the following data types.

OLE DB data types	ADO data type equivalent(s)
DBTYPE_BSTR	adBSTR
DBTYPE_BOOL	adBoolean
DBTYPE_DECIMAL	adDecimal
DBTYPE_UI1	adUnsignedTinyInt
DBTYPE_I1	adTinyInt
DBTYPE_UI2	adUnsignedSmallInt
DBTYPE_UI4	adUnsignedInt
DBTYPE_I8	adBigInt
DBTYPE_UI8	adUnsignedBigInt
DBTYPE_GUID	adGuid
DBTYPE_BYTES	adBinary, AdVarBinary, adLongVarBinary
DBTYPE_STR	adChar, adVarChar, adLongVarChar
DBTYPE_WSTR	adWChar, adVarWChar, adLongVarWChar
DBTYPE_NUMERIC	adNumeric

DBTYPE_DBDATE	adDBDate
DBTYPE_DBTIME	adDBTime
DBTYPE_DBTIMESTAMP	adDBTimeStamp
DBTYPE_VARNUMERIC	adVarNumeric
DBTYPE_FILETIME	adFileTime
DBTYPE_ERROR	adError

When the new field is of type decimal (in OLE DB, DBTYPE_DECIMAL, or in ADO, adDecimal), you must specify the precision and scale values.

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Issuing Commands to the Underlying Data Provider

Any command that does not begin with SHAPE is passed through to the data provider. This is equivalent to issuing a shape command in the form "SHAPE {provider command}". These commands do *not* have to produce a **Recordset**. For instance, "SHAPE {DROP TABLE MyTable}" is a perfectly valid shape command, assuming the data provider supports DROP TABLE.

This capability allows both normal provider commands and shape commands to share the same connection and transaction.

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Shape APPEND Clause

The shape command APPEND clause appends a column or columns to a **Recordset**. Often these columns are chapter columns, which refer to a [child Recordset](#).

Syntax

```
SHAPE [parent-command [[AS] parent-alias]] APPEND column-list
```

Description

The parts of this clause are as follows:

parent-command

Zero or one of the following (you may omit the *parent-command* entirely):

- A provider command within curly braces ("{}") that returns a **Recordset** object. The command is issued to the underlying [data provider](#), and its syntax depends on the requirements of that provider. This will typically be the SQL language, although ADO does not require any particular query language.
- Another shape command embedded in parentheses.
- The TABLE keyword, followed by the name of a table in the data provider.

parent-alias

An optional alias that refers to the parent **Recordset**.

column-list

One or more of the following:

- An aggregate column.
- A calculated column.
- A new column created with the NEW clause.
- A chapter column. A chapter column definition is enclosed in parentheses ("()"). See syntax below:

```
SHAPE [parent-command [[AS] parent-alias]]  
  APPEND (child-recordset [ [[AS] child-alias]  
    RELATE parent-column TO child-column | PARAMETER param-number,
```

[[AS] *chapter-alias*]
[, ...]

child-recordset

- A provider command within curly braces ("{}") that returns a **Recordset** object. The command is issued to the underlying [data provider](#), and its syntax depends on the requirements of that provider. This will typically be the SQL language, although ADO does not require any particular query language.
- Another shape command embedded in parentheses.
- The name of an existing shaped **Recordset**.
- The TABLE keyword, followed by the name of a table in the data provider.

child-alias

An alias that refers to the child **Recordset**.

parent-column

A column in the **Recordset** returned by the *parent-command*.

child-column

A column in the **Recordset** returned by the *child-command*.

param-number

See [Operation of Parameterized Commands](#).

chapter-alias

An alias that refers to the chapter column appended to the parent.

Note The "*parent-column TO child-column*" clause is actually a list, where each relation defined is separated by a comma.

Note The clause after the APPEND keyword is actually a list, where each clause is separated by a comma and defines another column to be appended to the parent.

Remarks

When you construct provider commands from user input as part of a SHAPE command, SHAPE will treat the user-supplied a provider command as an opaque string and pass them faithfully to the provider. For example, in the following SHAPE command,

```
SHAPE {select * from t1} APPEND ({select * from t2} RELATE k1 TO k2)
```

SHAPE will execute two commands: `select * from t1` and `(select * from t2 RELATE k1 TO k2)`. If the user supplies a compound command consisting of multiple provider commands separated by semicolons, SHAPE is not able to discern the difference. So in the following SHAPE command,

```
SHAPE {select * from t1; drop table t1} APPEND ({select * from t2} R
```

SHAPE executes `select * from t1; drop table t1` and `(select * from t2 RELATE k1 TO k2)`, not realizing that `drop table t1` is a separate and in this case, dangerous, provider command. Applications must always validate the user input to prevent such potential hacker attacks from happening.

Remarks

When you construct provider commands from user input as part of a SHAPE command, SHAPE will treat the user-supplied a provider command as an opaque string and pass them faithfully to the provider. For example, in the following SHAPE command,

```
SHAPE {select * from t1} APPEND ({select * from t2} RELATE k1 TO k2)
```

SHAPE will execute two commands: `select * from t1` and `(select * from t2 RELATE k1 TO k2)`. If the user supplies a compound command consisting of multiple provider commands separated by semicolons, SHAPE is not able to discern the difference. So in the following SHAPE command,

```
SHAPE {select * from t1; drop table t1} APPEND ({select * from t2} R
```

SHAPE executes `select * from t1; drop table t1` and `(select * from t2 RELATE k1 TO k2)`, not realizing that `drop table t1` is a separate and in this case, dangerous, provider command. Applications must always validate the user input to prevent such potential hacker attacks from happening.

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

ADO 2.5 

Operation of Non-Parameterized Commands

For non-parameterized commands, all of the provider commands are executed and the **Recordsets** are created during command execution. If the command is executed synchronously, all of the **Recordsets** will be fully populated. If an asynchronous population mode was selected, the populated state of the **Recordsets** will depend on the population mode and the size of the **Recordsets**.

For example, the *parent-command* could return a **Recordset** of customers for a company from a Customers table, and the *child-command* could return a **Recordset** of orders for all customers from an Orders table.

```
SHAPE {SELECT * FROM Customers}
  APPEND ({SELECT * FROM Orders} AS chapOrders
  RELATE customerID TO customerID)
```

For non-parameterized parent-child relationships, each parent and child **Recordset** object must have a column in common to associate them. The columns are named in the RELATE clause, *parent-column* first and then *child-column*. The columns may have different names in their respective **Recordset** objects but must refer to the same information in order to specify a meaningful relation. For example, the **Customers** and **Orders Recordset** objects could both have a customerID field. Because the membership of the child **Recordset** is determined by the provider command, it is possible for the child **Recordset** to contain orphaned rows. These orphaned rows are inaccessible without further reshaping.

Data shaping appends a chapter column to the parent **Recordset**. The values in the chapter column are references to rows in the child **Recordset**, which satisfy the RELATE clause. That is, the same value is in the *parent-column* of a given parent row as is in the *child-column* of all the rows of the chapter child. When multiple TO clauses are used in the same RELATE clause, they are implicitly combined using an AND operator. If the parent columns in the relate clause do not constitute a key to the parent **Recordset**, a single child row may have multiple parent rows.

When you access the reference in the chapter column, ADO automatically retrieves the **Recordset** represented by the reference. Note that in a non-parameterized command, although the entire child **Recordset** has been retrieved, the chapter only presents a subset of rows.

If the appended column has no *chapter-alias*, a name will be generated for it automatically. A [Field](#) object for the column will be appended to the **Recordset** object's [Fields](#) collection, and its data type will be **adChapter**.

For information about navigating a hierarchical **Recordset**, see [Accessing Rows in a Hierarchical Recordset](#).

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Operation of Parameterized Commands

If you are working with a large child **Recordset**, especially compared to the size of the parent **Recordset**, but need to access only a few child chapters, you might find it more efficient to use a parameterized command.

A *non-parameterized command* retrieves both the entire parent and child **Recordsets**, appends a chapter column to the parent, and then assigns a reference to the related child chapter for each parent row.

A *parameterized command* retrieves the entire parent **Recordset**, but retrieves only the chapter **Recordset** when the chapter column is accessed. This difference in retrieval strategy can yield significant performance benefits.

For example, you can specify the following:

```
SHAPE {SELECT * FROM customer}  
  APPEND ({SELECT * FROM orders WHERE cust_id = ?}  
  RELATE cust_id TO PARAMETER 0)
```

The parent and child tables have a column name in common, `cust_id`. The *child-command* has a "?" placeholder, to which the RELATE clause refers (that is, "...PARAMETER 0").

Note The PARAMETER clause pertains solely to the shape command syntax. It is not associated with either the ADO [Parameter](#) object or the [Parameters](#) collection.

When the parameterized shape command is executed, the following occurs:

1. The *parent-command* is executed and returns a parent **Recordset** from the Customers table.
2. A chapter column is appended to the parent **Recordset**.
3. When the chapter column of a parent row is accessed, the *child-command* is executed using the value of the `customer.cust_id` as the value of the parameter.

4. All rows in the data provider rowset created in step 3 are used to populate the child **Recordset**. In this example, that is all the rows in the Orders table in which the cust_id equals the value of customer.cust_id. By default, the child **Recordsets** will be cached on the client until all references to the parent **Recordset** are released. To change this behavior, set the **Recordset dynamic property Cache Child Rows** to **False**.
5. A reference to the retrieved child rows (that is, the chapter of the child **Recordset**) is placed in the chapter column of the current row of the parent **Recordset**.
6. Steps 3–5 are repeated when the chapter column of another row is accessed.

The **Cache Child Rows** dynamic property is set to **True** by default. The caching behavior varies depending upon the parameter values of the query. In a query with a single parameter, the child **Recordset** for a given parameter value will be cached between requests for a child with that value. The following code demonstrates this:

```

...
sCmd = "SHAPE {select * from customer} " & _
        "APPEND({select * from orders where cust_id = ?} " & _
        "RELATE cust_id TO PARAMETER 0) AS chpCustOrder"
Rst1.Open sCmd, Cnn1
Set RstChild = Rst1("chpCustOrder").Value
Rst1.MoveNext      ' Next cust_id passed to Param 0, & new rs fetched
                  ' into RstChild.
Rst1.MovePrevious  ' RstChild now holds cached rs, saving round trip
...

```

In a query with two or more parameters, a cached child is used only if all the parameter values match the cached values.

Parameterized Commands and Complex Parent Child Relations

In addition to using parameterized commands to improve performance of an equi-join type hierarchy, parameterized commands can be used to support more complex parent-child relationships. For example, consider a Little League database with two tables: one consisting of the teams (team_id, team_name) and the other of games (date, home_team, visiting_team).

Using a non-parameterized hierarchy, there is no way to relate the teams and games tables in such a way that the child **Recordset** for each team contains its

complete schedule. You can create chapters that contain the home schedule or the road schedule, but not both. This is because the RELATE clause limits you to parent-child relationships of the form (pc1=cc1) AND (pc2=pc2). So, if your command included "RELATE team_id TO home_team, team_id TO visiting_team", you would get only games where a team was playing itself. What you want is "(team_id=home_team) OR (team_id=visiting_team)", but the Shape provider does not support the OR clause.

To obtain the desired result, you can use a parameterized command. For example:

```
SHAPE {SELECT * FROM teams}
APPEND ({SELECT * FROM games WHERE home_team = ? OR visiting_team =
        RELATE team_id TO PARAMETER 0,
        team_id TO PARAMETER 1})
```

This example exploits the greater flexibility of the SQL WHERE clause to get the result you need.

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Hybrid Commands

Hybrid commands are partially parameterized commands. For example:

```
SHAPE {select * from plants}  
  APPEND( {select * from customers where country = ?}  
    RELATE PlantCountry TO PARAMETER 0,  
    PlantRegion TO CustomerRegion )
```

The caching behavior for a hybrid command is the same as that of regular parameterized commands.

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Intervening Shape COMPUTE Clauses

It is valid to embed one or more COMPUTE clauses between the parent and child in a parameterized shape command, as in the following example:

```
SHAPE {select au_lname, state from authors} APPEND
  ((SHAPE
    (SHAPE
      {select * from authors where state = ?} rs
      COMPUTE rs, ANY(rs.state) state, ANY(rs.au_lname) au_lname
      BY au_id) rs2
    COMPUTE rs2, ANY(rs2.state) BY au_lname)
  RELATE state TO PARAMETER 0)
```

See Also

[Data Shaping](#) | [Formal Shape Grammar](#) | [Shape Commands in General](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Shape COMPUTE Clause

A shape COMPUTE clause generates a [parent Recordset](#), whose columns consist of a reference to the [child Recordset](#); optional columns whose contents are chapter, new, or calculated columns, or the result of executing [aggregate functions](#) on the child **Recordset** or a previously shaped **Recordset**; and any columns from the child **Recordset** listed in the optional BY clause.

Syntax

```
SHAPE child-command [AS] child-alias  
    COMPUTE child-alias [[AS] name], [appended-column-list]  
    [BY grp-field-list]
```

Description

The parts of this clause are as follows:

child-command

Consists of one of the following:

- A query command within curly braces ("{}") that returns a child **Recordset** object. The command is issued to the underlying [data provider](#), and its syntax depends on the requirements of that provider. This will typically be the SQL language, although ADO does not require any particular query language.
- The name of an existing shaped **Recordset**.
- Another shape command.
- The TABLE keyword, followed by the name of a table in the data provider.

child-alias

An alias used to refer to the **Recordset** returned by the *child-command*. The *child-alias* is required in the list of columns in the COMPUTE clause and defines the relation between the parent and child **Recordset** objects.

appended-column-list

A list in which each element defines a column in the generated parent. Each element contains either a chapter column, a new column, a calculated

column, or a value resulting from an [aggregate function](#) on the child **Recordset**.

grp-field-list

A list of columns in the parent and child **Recordset** objects that specifies how rows should be grouped in the child.

For each column in the *grp-field-list*, there is a corresponding column in the child and parent **Recordset** objects. For each row in the parent **Recordset**, the *grp-field-list* columns have unique values, and the child **Recordset** referenced by the parent row consists solely of child rows whose *grp-field-list* columns have the same values as the parent row.

If the BY clause is included, the child **Recordset**'s rows will be grouped based on the columns in the COMPUTE clause. The parent **Recordset** will contain one row for each group of rows in the child **Recordset**.

If the BY clause is omitted, the entire child **Recordset** is treated as a single group and the parent **Recordset** will contain exactly one row. That row will reference the entire child **Recordset**. Omitting the BY clause allows you to compute "grand total" aggregates over the entire child **Recordset**.

For example:

```
SHAPE {select * from Orders} AS orders
  COMPUTE orders, SUM(orders.OrderAmount) as TotalSales
```

Regardless of which way the parent **Recordset** is formed (using COMPUTE or using APPEND), it will contain a chapter column that is used to relate it to a child **Recordset**. If you wish, the parent **Recordset** may also contain columns that contain aggregates (SUM, MIN, MAX, and so on) over the child rows. Both the parent and the child **Recordset** may contain columns that contain an expression on the row in the **Recordset**, as well as columns that are new and initially empty.

Operation

The *child-command* is issued to the provider, which returns a child **Recordset**.

The COMPUTE clause specifies the columns of the parent **Recordset**, which may be a reference to the child **Recordset**, one or more aggregates, a [calculated](#)

[expression](#), or new columns. If there is a BY clause, the columns it defines are also appended to the parent **Recordset**. The BY clause specifies how the rows of the child **Recordset** are grouped.

For example, assume you have a table—Demographics—consisting of State, City, and Population fields (the population figures are solely for illustration).

State	City	Population
WA	Seattle	700,000
OR	Medford	200,000
OR	Portland	400,000
CA	Los Angeles	800,000
CA	San Diego	600,000
WA	Tacoma	500,000
OR	Corvallis	300,000

Now, issue this shape command:

```
rst.Open "SHAPE {select * from demographics} AS rs " & _  
        "COMPUTE rs, SUM(rs.population) BY state", _  
        objConnection
```

This command opens a shaped **Recordset** with two levels. The parent level is a generated **Recordset** with an aggregate column (SUM(rs.population)), a column referencing the child **Recordset** (rs), and a column for grouping the child **Recordset** (state). The child level is the **Recordset** returned by the query command (select * from demographics).

The child **Recordset** detail rows will be grouped by state, but otherwise in no particular order. That is, the groups will not be in alphabetical or numerical order. If you want the parent **Recordset** to be ordered, you can use the **Recordset Sort** method to order the parent **Recordset**.

You can now navigate the opened parent **Recordset** and access the child detail **Recordset** objects. For more information, see [Accessing Rows in a Hierarchical Recordset](#).

Resultant Parent and Child Detail Recordsets

Parent

SUM (rs.Population)	rs	State
1,300,000	Reference to child1	CA
1,200,000	Reference to child2	WA
1,100,000	Reference to child3	OR

Child1

State	City	Population
CA	Los Angeles	800,000
CA	San Diego	600,000

Child2

State	City	Population
WA	Seattle	700,000
WA	Tacoma	500,000

Child3

State	City	Population
OR	Medford	200,000
OR	Portland	400,000
OR	Corvallis	300,000

See Also

[Accessing Rows in a Hierarchical Recordset](#) | [Data Shaping Summary](#) | [Field Object](#) | [Formal Shape Grammar](#) | [Recordset Object](#) | [Required Providers for Data Shaping](#) | [Shape APPEND Clause](#) | [Shape Commands in General](#) | [Value Property](#) | [Visual Basic for Applications Functions](#)

ADO 2.5 

Fabricating Hierarchical Recordsets

The following example shows how to fabricate a [hierarchical Recordset](#) without an underlying data source by using the data shaping grammar to define columns for parent, child, and grandchild **Recordsets**.

To fabricate a hierarchical **Recordset**, you must specify the Microsoft Data Shaping Service for OLE DB (MSDataShape), and you may specify a Data Provider value of NONE in the connection string parameter of the [Connection](#) object's [Open](#) method. For more information, see [Required Providers for Data Shaping](#).

```
Dim cn As New ADODB.Connection
Dim rsCustomers As New ADODB.Recordset

cn.Open "Provider=MSDataShape;Data Provider=NONE;"

strShape = _
"SHAPE APPEND NEW adInteger AS CustID," & _
    " NEW adChar(25) AS FirstName," & _
    " NEW adChar(25) AS LastName," & _
    " NEW adChar(12) AS SSN," & _
    " NEW adChar(50) AS Address," & _
    " ((SHAPE APPEND NEW adChar(80) AS VIN_NO," & _
        " NEW adInteger AS CustID," & _
        " NEW adChar(20) AS BodyColor," & _
        " ((SHAPE APPEND NEW adChar(80) AS VIN_NO," & _
            " NEW adChar(20) AS Make," & _
            " NEW adChar(20) AS Model," & _
            " NEW adChar(4) AS Year)" & _
        " AS VINS RELATE VIN_NO TO VIN_NO))" & _
    " AS Vehicles RELATE CustID TO CustID) "
```

```
rsCustomers.Open strShape, cn, adOpenStatic, adLockOptimistic, -1
```

Once the **Recordset** has been fabricated, it can be populated, manipulated, or persisted to a file.

See Also

[Accessing Rows in a Hierarchical Recordset](#) | [Formal Shape Grammar](#) |

[Required Providers for Data Shaping](#) | [Shape APPEND Clause](#) | [Shape Commands in General](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Accessing Rows in a Hierarchical Recordset

The following example shows the steps necessary to access rows in a hierarchical [Recordset](#):

1. **Recordset** objects from the authors and titleauthor tables are related by author ID.
2. The outer loop displays each author's first and last name, state, and identification.
3. The appended **Recordset** for each row is retrieved from the **Fields** collection and assigned to *rstTitleAuthor*.
4. The inner loop displays four fields from each row in the appended **Recordset**.

(The [StayInSync](#) property is set to FALSE for purposes of illustration—so you can see the chapter change explicitly in each iteration of the outer loop. However, the example will be more efficient if the assignment in step 3 is moved before the first line in step 2, so that the assignment is performed only once. Then set the **StayInSync** property to TRUE, so that *rstTitleAuthor* will implicitly and automatically change to the corresponding chapter whenever *rst* moves to a new row.)

Example

```
Sub datashape()  
    Dim cnn As New ADODB.Connection  
    Dim rst As New ADODB.Recordset  
    Dim rstTitleAuthor As New ADODB.Recordset  
  
    cnn.Provider = "MSDataShape"  
    cnn.Open     "Data Provider=MSDASQL;" & _  
                "Data Source=SRV;" & _  
                "User Id=MyUserName;Password=MyPassword;Database=Pubs  
' STEP 1  
    rst.StayInSync = FALSE  
    rst.Open     "SHAPE {select * from authors} " & _  
                "APPEND ({select * from titleauthor} " & _  
                "RELATE au_id TO au_id) AS chapTitleAuthor", _
```

```
        cnn
' STEP 2
  While Not rst.EOF
    Debug.Print    rst("au_fname"), rst("au_lname"), _
                  rst("state"), rst("au_id")
' STEP 3
  Set rstTitleAuthor = rst("chapTitleAuthor").Value
' STEP 4
  While Not rstTitleAuthor.EOF
    Debug.Print rstTitleAuthor(0), rstTitleAuthor(1), _
              rstTitleAuthor(2), rstTitleAuthor(3)
    rstTitleAuthor.MoveNext
  Wend
  rst.MoveNext
Wend
End Sub
```

See Also

[Data Shaping Summary](#) | [Field Object](#) | [Fields Collection](#) | [Formal Shape Grammar](#) | [Microsoft Data Shaping Service for OLE DB](#) | [Recordset Object](#) | [Required Providers for Data Shaping](#) | [Shape APPEND Clause](#) | [Shape Commands in General](#) | [Shape COMPUTE Clause](#) | [Visual Basic for Applications Functions](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Formal Shape Grammar

This is the formal grammar for creating any shape command:

- Required grammatical terms are text strings delimited by angle brackets ("`<>`").
- Optional terms are delimited by square brackets ("`[]`").
- Alternatives are indicated by a virgule ("`|`").
- Repeating alternatives are indicated by an ellipsis ("`...`").
- *Alpha* indicates a string of alphabetical letters.
- *Digit* indicates a string of numbers.
- *Unicode-digit* indicates a string of unicode digits.

All other terms are literals.

Term	Definition
<code><shape-command></code>	<code>SHAPE [<code><table-exp></code> [[<code>AS</code>] <code><alias></code>]][<code><shape-act</code> <code>{<provider-command-text>} </code></code>
<code><table-exp></code>	<code>(<shape-command>) </code> <code>TABLE <code><quoted-name></code> </code> <code><code><quoted-name></code></code>
<code><shape-action></code>	<code>APPEND <code><aliased-field-list></code> </code> <code>COMPUTE <code><aliased-field-list></code> [<code>BY</code> <code><field-list></code></code>
<code><aliased-field-list></code>	<code><aliased-field></code> [<code>,</code> <code><aliased-field...></code>]
<code><aliased-field></code>	<code><field-exp></code> [[<code>AS</code>] <code><alias></code>]
<code><field-exp></code>	<code>(<relation-exp>) </code> <code><calculated-exp> </code> <code><aggregate-exp> </code> <code><new-exp></code>

<relation_exp>	<table-exp> [[AS] <alias>] RELATE <relation-cond-list>
<relation-cond-list>	<relation-cond> [, <relation-cond>...]
<relation-cond>	<field-name> TO <child-ref>
<child-ref>	<field-name> PARAMETER <param-ref>
<param-ref>	<number>
<field-list>	<field-name> [, <field-name>]
<aggregate-exp>	SUM(<qualified-field-name>) AVG(<qualified-field-name>) MIN(<qualified-field-name>) MAX(<qualified-field-name>) COUNT(<qualified-alias> <qualified-name>) STDEV(<qualified-field-name>) ANY(<qualified-field-name>)
<calculated-exp>	CALC(<expression>)
<qualified-field-name>	<alias>.[<alias>...]<field-name>
<alias>	<quoted-name>
<field-name>	<quoted-name> [[AS] <alias>]
<quoted-name>	"<string>" '<string>' [<string>]

<name>	<name>
<qualified-name>	alias[.alias...]
<name>	alpha [alpha digit _ # : ...]
<number>	digit [digit...]
<new-exp>	NEW <field-type> [(<number> [, <number>])]
<field-type>	An OLE DB or ADO data type.
<string>	unicode-char [unicode-char...]
<expression>	A Visual Basic for Applications expression whose operands are other non-CALC columns in the same row.

See Also

[Accessing Rows in a Hierarchical Recordset](#) | [Data Shaping Summary](#) | [Required Providers for Data Shaping](#) | [Shape APPEND Clause](#) | [Shape Commands in General](#) | [Shape COMPUTE Clause](#) | [Visual Basic for Applications Functions](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

Visual Basic for Applications Functions

The following Visual Basic for Applications functions can be used in data shaping CALC expressions:

Abs	Asc	Atn	CBool	CByte	CCur
CDate	CDbl	Chr	ChrB	ChrW	Chr\$
ChrB\$	CInt	CLng	Cos	CSng	CStr
Cvar	CVDate	CVErr	Date	Date\$	DateAdd
DateDiff	DatePart	DateSerial	DateValue	Day	DDB
Error	Error\$	Exp	Fix	Format	Format\$
FV	Hex	Hex\$	Hour	IIF	InStr
Int	IPmt	IRR	IsDate	IsEmpty	IsError
IsNull	IsNumeric	IsObject	LCase	LCase\$	Left
LeftB	Left\$	LeftB\$	Len	Log	LTrim
LTrim\$	Mid	Mid\$	Minute	MIRR	Month
Now	NPer	NPV	Oct	Oct\$	Pmt
PPmt	PV	QBColor	Rate	RGB	Right
RightB	Right\$	RightB\$	Rnd	RTrim	RTrim\$
Second	Sgn	Sin	SLN	Space	Space\$
Sqr	Str	Str\$	StrComp	StrConv	String
String\$	SYD	Tan	Time	Time\$	Timer
TimeSerial	TimeValue	Trim	Trim\$	TypeName	UCase
UCase\$	Val	VarType	Weekday	Year	

See Also

[Accessing Rows in a Hierarchical Recordset](#) | [Data Shaping Summary](#) | [Formal Shape Grammar](#) | [Required Providers for Data Shaping](#) | [Shape APPEND Clause](#) | [Shape Commands in General](#) | [Shape COMPUTE Clause](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

Chapter 10: Records and Streams

ADO currently provides the [Recordset](#) object as the primary means of accessing information in data sources, such as relational databases. However, some providers support the [Record](#) and [Stream](#) objects as alternative or complementary objects with which data from providers can be manipulated. For specifics on **Record** behavior, see your provider's documentation.

Records

Record objects essentially function as one-row **Recordsets**. However, **Records** have limited functionality compared to **Recordsets** and they have different properties and methods. The source for the data in a **Record** object can be a command which returns one row of data from the provider. Using **Record** objects rather than **Recordset** objects to receive the results from a query that returns one row of data eliminates the overhead of instantiating the more complex **Recordset** object.

Record objects can serve another purpose, particularly with providers for data sources other than traditional relational databases, such as the [Microsoft OLE DB Provider for Internet Publishing](#). Much of the information that must be processed exists, not as tables in databases, but as messages in electronic mail systems and files in modern file systems. The **Record** and **Stream** objects facilitate access to information stored in sources other than relational databases.

The **Record** object can represent and manage data such as directories and files in a file system or folders and messages in an e-mail system. For these purposes, the source for the **Record** can be the current row of an open **Recordset**, an absolute URL, or a relative URL in conjunction with an open [Connection](#) object.

Typically, a **Recordset** can be used to represent a container or parent in a hierarchy such as a folder or directory. A **Record** can be used to return specific information about one node in the parent container, such as a file or document. The primary reason **Records** are used to represent this type of information is that these sources of data are heterogenous. This means that each **Record** may have a different set and number of fields. Traditional **Recordsets** containing rows from a database are homogenous, which means that each row has the same number and type of fields.

For more information about using the **Record** object for processing this heterogeneous data from providers such as the Internet Publishing Provider, see [Using ADO for Internet Publishing](#).

Streams

The **Stream** object provides the means to read, write, and manage a stream of bytes. This byte stream may be text or binary and is limited in size only by system resources. Typically, ADO **Stream** objects are used for the following purposes:

- To contain the text or bytes that comprise a file or message, typically used with providers such as the Microsoft OLE DB Provider for Internet Publishing. For more information about this use of **Stream** objects, see [Using ADO for Internet Publishing](#).

A **Stream** object can be opened on:

- A simple file specified with a URL.
- A field of a **Record** or **Recordset** containing a **Stream** object.
- The default stream of a **Record** or **Recordset** object representing a directory or compound file.
- A resource field containing the URL of a simple file.
- No particular source at all. In this case, a **Stream** object is opened in memory. Data can be written to it and then saved in another **Stream** or file.
- A BLOB field in a **Recordset**.

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Streams and Persistence

The [Recordset](#) object [Save](#) method stores, or *persists*, a **Recordset** in a file, and the [Open](#) method restores the **Recordset** from that file.

With ADO 2.5, the **Save** and **Open** methods can persist a **Recordset** to a [Stream](#) object as well. This feature is especially useful when working with Remote Data Service (RDS) and Active Server Pages (ASP).

For more information about how persistence can be used by itself on ASP pages, see the current ASP documentation.

The following are a few scenarios that show how **Stream** objects and persistence can be used.

Scenario 1

This scenario simply saves a **Recordset** to a file and then to a **Stream**. It then opens the persisted stream into another **Recordset**.

```
Dim rs1 As ADODB.Recordset
Dim rs2 As ADODB.Recordset
Dim stm As ADODB.Stream

Set rs1 = New ADODB.Recordset
Set rs2 = New ADODB.Recordset
Set stm = New ADODB.Stream

rs1.Open "SELECT * FROM Customers", "Provider=sqloledb;" & _
        "Data Source=MyServer;Initial Catalog=Northwind;" & _
        "Integrated Security=SSPI;""", adopenStatic, adLockReadOnly,
rs1.Save "c:\myfolder\mysubfolder\myrs.xml", adPersistXML
rs1.Save stm, adPersistXML
rs2.Open stm
```

Scenario 2

This scenario persists a **Recordset** into a **Stream** in XML format. It then reads the **Stream** into a string that you can examine, manipulate, or display.

```
Dim rs As ADODB.Recordset
Dim stm As ADODB.Stream
Dim strRst As String

Set rs = New ADODB.Recordset
Set stm = New ADODB.Stream

' Open, save, and close the recordset.
rs.Open "SELECT * FROM Customers", "Provider=sqloledb;" & _
        "Data Source=MyServer;Initial Catalog=Northwind;" & _
        "Integrated Security=SSPI;""
rs.Save stm, adPersistXML
rs.Close
Set rs = nothing

' Put saved Recordset into a string variable.
strRst = stm.ReadText(adReadAll)

' Examine, manipulate, or display the XML data.
...
```

Scenario 3

This example code shows ASP code persisting a **Recordset** as XML directly to the **Response** object:

```
...
<%
response.ContentType = "text/xml"

' Create and open a Recordset.
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open "select * from Customers", "Provider=sqloledb;" & _
        "Data Source=MyServer;Initial Catalog=Northwind;" & _
        "Integrated Security=SSPI;""

' Save Recordset directly into output stream.
rs.Save Response, adPersistXML

' Close Recordset.
rs.Close
Set rs = nothing
%>
...
```

Scenario 4

In this scenario, ASP code writes the contents of the **Recordset** in ADTG format to the client. The [Microsoft Cursor Service for OLE DB](#) can use this data to create a disconnected **Recordset**.

A new property on the RDS [DataControl](#), [URL](#), points to the .asp page that generates the **Recordset**. This means a **Recordset** object can be obtained without RDS using the server-side [DataFactory](#) object or the user writing a business object. This simplifies the RDS programming model significantly.

Server-side code, named `http://server/directory/recordset.asp`:

```
<%
Dim rs
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open "select au_fname, au_lname, phone from Authors", "" & _
        "Provider=sqloledb;Data Source=MyServer;" & _
        "Initial Catalog=Pubs;Integrated Security=SSPI;"
response.ContentType = "multipart/mixed"
rs.Save response, adPersistADTG
%>
```

Client-side code:

```
<HTML>
<HEAD>
<TITLE>RDS Query Page</TITLE>
</HEAD>
<body>
<CENTER>
<H1>Remote Data Service 2.5</H1>
<TABLE DATASRC="#DC1">
  <TR>
    <TD><SPAN DATAFLD="au_fname"></SPAN></TD>
    <TD><SPAN DATAFLD="au_lname"></SPAN></TD>
    <TD><SPAN DATAFLD="phone"></SPAN></TD>
  </TR>
</TABLE>
<BR>

<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID=DC1 HEIGHT=1 WIDTH = 1>
  <PARAM NAME="URL" VALUE="http://server/directory/recordset.asp">
```

```
</OBJECT>
```

```
</SCRIPT>
```

```
</BODY>
```

```
</HTML>
```

Developers also have the option of using a **Recordset** object on the client:

```
...  
function GetRs()  
  {  
    rs = CreateObject("ADODB.Recordset");  
    rs.Open "http://server/directory/recordset.asp"  
    DC1.SourceRecordset = rs;  
  }  
...
```

See Also

[Open Method \(ADO Recordset\)](#) | [Record Object](#) | [Save Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Using ADO for Internet Publishing

[The OLE DB Provider for Internet Publishing](#) shows a specific example of accessing heterogeneous data with ADO. While the examples in this section will be specific to using the Internet Publishing Provider, the principles demonstrated should be similar when using ADO with other providers to heterogeneous data, such as a provider to an e-mail store.

URLs

Uniform Resource Locators (URLs) can be used as an alternative to connection strings and command text to specify data sources and the location of files and directories. You can use URLs with the existing [Connection](#) and **Recordset** objects as well as with the **Record** and **Stream** objects.

For more information about using URLs, see [Absolute and Relative URLs](#).

Record Fields

The distinguishing difference between heterogeneous data and homogeneous data is that for the former, each row of data, or **Record**, can have a different set of columns, or **Fields**. For homogeneous data, each row has the same set of columns. For more information about the fields specific to the Internet Publishing Provider, see [Records and Provider-Supplied Extra Fields](#).

Appending New Fields

Several ADO objects have been enhanced to work in conjunction with **Record** and **Stream** objects.

- The [Fields](#) collection [Append](#) method, which creates and adds a [Field](#) object to the collection, can also specify the value of the **Field**.
- The [Update](#) method finalizes the addition or deletion of fields to the collection.
- As a shortcut and alternative to the **Append** method, you may create fields by simply assigning a value to an undefined or previously deleted field.

See Also

[Record Object](#) | [Stream Object](#) | [What's New in ADO](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

The OLE DB Provider for Internet Publishing

The ADO [Record](#) and [Stream](#) objects can be used with the Microsoft OLE DB Provider for Internet Publishing (Internet Publishing Provider) to access and manipulate resources, such as Web folders or files served by Microsoft FrontPage. With ADO, you can specify the source of a **Record**, **Stream**, or [Recordset](#) to be a URL. You can then upload, download, move, copy, and delete resources, or directly manipulate resource properties.

For example code using **Records** and **Streams** with the Internet Publishing Provider, see the [Internet Publishing Scenario](#).

The Internet Publishing Provider is installed with Microsoft Windows 2000. Earlier versions of the Internet Publishing Provider are also available with Microsoft Office 2000 and Microsoft Internet Explorer 5.0.

There are three ways to connect ADO to the Internet Publishing Provider:

- Specify "URL=" in the connection string. For example:

```
objConn.Open "URL=http://servername"
```

- Specify Msdaipp.dso for the *Provider* keyword of the connection string. For example:

```
objConn.Open "provider=MSDAIPP.DSO;data source=http://servername"
```

- Specify Msdaipp.dso for the [Provider](#) property of the [Connection](#) object. For example:

```
objConn.Provider = "MSDAIPP.DSO"  
objConn.Open "http://servername"
```

Note If Msdaipp.dso is explicitly specified as the value of the provider, either with the *Provider* connection string keyword or the **Provider** property, you cannot use "URL=" in the connection string. If you do, an error will occur. Instead, simply specify the URL as shown above.

For more specific information about the Internet Publishing Provider, see [Microsoft OLE DB Provider for Internet Publishing](#), or the provider documentation provided with the source application with which the OLE DB Provider for Internet Publishing was installed: Windows 2000, Office 2000, or Internet Explorer 5.0.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Internet Publishing Scenario

This code example demonstrates how to use ADO with the Microsoft OLE DB Provider for Internet Publishing. In this scenario, you will create a Visual Basic application that uses **Recordset**, **Record**, and **Stream** objects to display the contents of resources published with the Internet Publishing Provider.

The following steps are necessary to create this scenario:

- [Step 1: Set Up the Visual Basic Project](#)
- [Step 2: Initialize the Main List Box](#)
- [Step 3: Populate the Fields List Box](#)
- [Step 4: Populate the Details Text Box](#)

See Also

[Microsoft OLE DB Provider for Internet Publishing](#) | [The OLE DB Provider for Internet Publishing](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Step 1: Set Up the Visual Basic Project

In this scenario, it is assumed that you have Microsoft Visual Basic 6.0 or later, ADO 2.5 or later, and the Microsoft OLE DB Provider for Internet Publishing installed on your system.

To create an ADO project

1. In Microsoft Visual Basic, create a new Standard EXE project.
2. From the **Project** menu, choose **References**.
3. Select "**Microsoft ActiveX Data Objects 2.5 Library**" and click **OK**.

To insert controls on the main form

1. Add a ListBox control to Form1. Set its **Name** property to lstMain.
2. Add another ListBox control to Form1. Set its **Name** property to lstDetails.
3. Add a TextBox control to Form1. Set its **Name** property to txtDetails.

See Also

[Step 2: Initialize the Main List Box](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Step 2: Initialize the Main List Box

To declare global Record and Recordset objects

- Insert the following code into the (General) (Declarations) for Form1:

```
Option Explicit
Dim grec As Record
Dim grs As Recordset
```

This code declares global object references for **Record** and **Recordset** objects that will be used later in this scenario.

To connect to a URL and populate lstMain

- Insert the following code into the Form Load event handler for Form1:

```
Private Sub Form_Load()
    Set grec = New Record
    Set grs = New Recordset
    grec.Open "", "URL=http://servername/foldername/", , _
        adOpenIfExists Or adCreateCollection
    Set grs = grec.GetChildren
    While Not grs.EOF
        lstMain.AddItem grs(0)
        grs.MoveNext
    Wend
End Sub
```

This code instantiates the global **Record** and **Recordset** objects. The **Record**, grec, is opened with a URL specified as the **ActiveConnection**. If the URL exists, it is opened; if it does not already exist, it is created. Note that you should replace "http://servername/foldername/" with a valid URL from your environment.

The **Recordset**, grs, is opened on the children of the **Record**, grec. Then lstMain is populated with the file names of the resources published to the URL.

See Also

Step 3: Populate the Fields List Box

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 

Step 3: Populate the Fields List Box

To populate the Fields list box

Insert the following code into the Click event handler of lstMain:

```
Private Sub lstMain_Click()  
    Dim rec As Record  
    Dim rs As Recordset  
    Set rec = New Record  
    Set rs = New Recordset  
    grs.MoveFirst  
    grs.Move lstMain.ListIndex  
    lstDetails.Clear  
    rec.Open grs  
    Select Case rec.RecordType  
        Case adCollectionRecord:  
            Set rs = rec.GetChildren  
            While Not rs.EOF  
                lstDetails.AddItem rs(0)  
                rs.MoveNext  
            Wend  
        Case adSimpleRecord:  
            recFields rec, lstDetails, txtDetails  
    End Select  
End Sub
```

This code declares and instantiates local **Record** and **Recordset** objects, `rec` and `rs`, respectively.

The row corresponding to the resource selected in `lstMain` is made the current row of `grs`. Then the **Details** list box is cleared and `rec` is opened with the current row of `grs` as the source.

If the resource is a collection record (as specified by **RecordType**), the local **Recordset**, `rs`, is opened on the children of `rec`. Then `lstDetails` is filled with the values from the rows of `rs`.

If the resource is a simple record, `recFields` is called. For more information about `recFields`, see the next step.

No code is implemented if the resource is a structured document.

See Also

[Step 4: Populate the Details Text Box](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Step 4: Populate the Details Text Box

To populate the Details text box

Create a new subroutine named recFields and insert the following code:

```
Sub recFields(r As Record, l As ListBox, t As TextBox)
    Dim f As Field
    Dim s As Stream
    Set s = New Stream
    Dim str As String

    For Each f In r.Fields
        l.AddItem f.Name & ": " & f.Value
    Next
    t.Text = ""
    If r!RESOURCE_CONTENTCLASS = "text/plain" Then
        s.Open r, adModeRead, adOpenStreamFromRecord
        str = s.ReadText(1)
        s.Position = 0
        If Asc(Mid(str, 1, 1)) = 63 Then '//63 = "?"
            s.Charset = "ascii"
            s.Type = adTypeText
        End If
        t.Text = s.ReadText(adReadAll)
    End If
End Sub
```

This code populates lstDetails with the fields and values of the simple record passed to recFields. If the resource is a text file, a text **Stream** is opened from the resource record. The code determines if the character set is ASCII and copies the **Stream** contents into txtDetails.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Absolute and Relative URLs

A URL specifies the location of a target stored on a local or networked computer, such as a file, directory, HTML page, image, program, and so on. In this discussion, an *absolute URL* is of the form:

scheme://server/path/resource

where:

scheme

Specifies how the *resource* is to be accessed.

server

Specifies the name of the computer where the *resource* is located.

path

Specifies the sequence of directories leading to the target. If *resource* is omitted, the target is the last directory in *path*.

resource

If included, *resource* is the target, and is typically the name of a file. It may be a *simple file*, containing a single binary stream of bytes, or a *structured document*, containing one or more storages and binary streams of bytes.

An *absolute URL* contains all the information necessary to locate a resource.

A *relative URL* locates a resource using an absolute URL as a starting point. In effect, the "complete URL" of the target is specified by concatenating the absolute and relative URLs. A relative URL typically consists only of the *path*, and optionally, the *resource*, but no *scheme* or *server*.

URL Scheme Registration

If a provider supports URLs, it will register for one or more URL schemes. This means that any URLs using this scheme will automatically invoke the registered provider. For example, the *http* scheme is registered to the [Microsoft OLE DB Provider for Internet Publishing](#). ADO assumes all URLs prefixed with "http" represent Web folders or files to be used with the Internet Publishing Provider. For information about the schemes registered by your provider, see your provider documentation.

Defining Context with a URL

One function of an open connection, represented by a [Connection](#) object, is to restrict subsequent operations to the data source represented by that connection. That is, the connection defines the context for subsequent operations.

With ADO 2.5, an absolute URL may also define a context. For example, when a [Record](#) object is opened with an absolute URL, a **Connection** object is implicitly created to represent the resource specified by the URL.

An absolute URL that defines a context may be specified in the *ActiveConnection* parameter of the **Record** object [Open](#) method. An absolute URL may also be specified as the value of the new "URL=" keyword in the **Connection** object [Open](#) method *ConnectionString* parameter, and the [Recordset](#) object [Open](#) method *ActiveConnection* parameter.

Context may also be defined with an open **Record** or **Recordset** object that represents a directory because these objects already have an implicitly or explicitly declared **Connection** object that specifies context.

Scoped Operations

The context simultaneously defines a *scope*—that is, the directory and its subdirectories that may participate in subsequent operations. The **Record** object has several scoped methods, including [CopyRecord](#), [MoveRecord](#), and [DeleteRecord](#), that operate on a directory and all its subdirectories.

Relative URLs as Command Text

A string specifying a command to be executed on the data source may be specified in the **Connection** object [Execute](#) method *CommandText* parameter, and the **Recordset** object **Open** method *Source* parameter.

A relative URL may be specified in the *CommandText* or *Source* parameter. The relative URL does not actually specify a command (such as an SQL command); it is merely specified in those parameters. In addition, the context of the active connection must be an absolute URL, and the *Option* parameter must be set to **adCmdTableDirect**.

For example, a **Recordset** could be opened on the Readme25.txt file of the Winnt/system32 directory like this:

```
recordset.Open "system32/Readme25.txt", "URL=http://YourServer/Winnt
```

The absolute URL in the connection string specifies the server (YourServer) and the path (winnt). This URL also defines the context.

The relative URL in the command text uses the absolute URL as a starting point and specifies the remainder of the path (system32) and the file to open (Readme25.txt).

The options field (adCmdTableDirect) indicates that the command type is a relative URL.

As another example, the following code will open a **Recordset** on the contents of the winnt directory:

```
recordset.Open "", "URL=http://YourServer/Winnt/", , , adCmdTableDirect
```

OLE DB Provider-Supplied URL Schemes

The leading part of a fully-qualified URL is the *scheme* used to access the resource identified by the remainder of the URL. Examples are HTTP (HyperText Transfer Protocol) and FTP (File Transfer Protocol).

ADO supports OLE DB providers that recognize their own URL schemes. For example, the [Microsoft OLE DB Provider for Internet Publishing](#), which accesses "published" Windows 2000 files, recognizes the existing HTTP scheme.

See Also

[Connection Object](#) | [Record Object](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Records and Provider-Supplied Fields

When a [Record](#) object is opened, its source can be the current row of an open [Recordset](#), an absolute URL, or a relative URL in conjunction with an open [Connection](#) object.

If the **Record** is opened from a **Recordset**, the **Record** object [Fields](#) collection will contain all the fields from the **Recordset**, plus any fields added by the underlying provider.

The provider may insert additional fields that serve as supplementary characteristics of the **Record**. As a result, a **Record** may have unique fields not in the **Recordset** as a whole or any **Record** derived from another row of the **Recordset**.

For example, all rows of a **Recordset** derived from an e-mail data source might have columns such as From, To, and Subject. A **Record** derived from that **Recordset** will have the same fields. However, the **Record** may also have other fields unique to the particular message represented by that **Record**, such as Attachment and Cc (carbon copy).

Although the **Record** object and current row of the **Recordset** have the same fields, they are different because **Record** and **Recordset** objects have different methods and properties.

A field held in common by the **Record** and **Recordset** can be modified on either object. However, the field cannot be deleted on the **Record** object, although the underlying provider may support setting the field to null.

After the **Record** is opened, you can programmatically add fields. You can also delete fields you have added, but you cannot delete fields from the original **Recordset**.

You may also open the **Record** object directly from a URL. In this case, the fields added to the **Record** depend on the underlying provider. Currently, most providers add a set of fields that describe the entity represented by the **Record**. If the entity consists of a stream of bytes, such as a simple file, then a [Stream](#)

object can usually be opened from the **Record**.

Special Fields for Document Source Providers

A special class of providers, called *document source providers*, manages folders and documents. When a **Record** object represents a document or a **Recordset** object represents a folder of documents, the document source provider populates those objects with a unique set of fields that describe characteristics of the document instead of the actual document itself. Typically, one field contains a reference to the **Stream** that represents the document.

These fields constitute a resource **record** or **recordset** and are listed for the specific providers that support them in [Appendix A: Providers](#).

Two constants index the **Fields** collection of a resource **Record** or **Recordset** to retrieve a pair of commonly used fields. The **Field** object [Value](#) property returns the desired content.

- The field accessed with the **adDefaultStream** constant contains a default stream associated with the **Record** or **Recordset** object. The provider assigns a default stream to an object.
- The field accessed with the **adRecordURL** constant contains the absolute URL that identifies the document.

A document source provider does not support the [Properties](#) collection of **Record** and **Field** objects. The content of the **Properties** collection is null for such objects.

A document source provider may add a provider-specific property such as **Datasource Type** to identify whether it is a document source provider. For more information about how to determine your type of provider, see your provider documentation.

Resource Recordset Columns

A *resource recordset* consists of the following columns.

Column name	Type	Descrip
RESOURCE_PARSENAME	AdVarChar	Read-only. Ind URL of the res
RESOURCE_PARENTNAME	AdVarChar	Read-only. Ind absolute URL of parent record.
RESOURCE_ABSOLUTEPARSENAME	AdVarChar	Read-only. Ind absolute URL of resource, which concatenation of PARENTNAME and PARSENAME
RESOURCE_ISHIDDEN	AdBoolean	True if the resource is hidden. No rows returned unless command that rowset explicit rows where RESOURCE_ISHIDDEN is True.
RESOURCE_ISREADONLY	AdBoolean	True if the resource is read-only. Attempt to open this resource with DBBINDFLAG_DB_E_READWRITE and will fail with DB_E_READONLY. This property is read-only and cannot be edited even when the resource has not been opened for read-write. Indicates the link to the document.

RESOURCE_CONTENTTYPE	AdVarWChar	example, a law This may corre the Office tem to create the d Indicates the M of the documen indicating the such as "text/ Indicates the la which the cont stored.
RESOURCE_CONTENTCLASS	AdVarWChar	Read-only. Ind FILETIME str containing the resource was c time is reporte Coordinated U Time (UTC) fo Read-only. Ind FILETIME str containing the the resource w accessed. The UTC format. T FILETIME me zero if the prov not support thi member.
RESOURCE_CONTENTLANGUAGE	AdVarWChar	Read-only. Ind FILETIME str containing the resource was c time is reporte Coordinated U Time (UTC) fo Read-only. Ind FILETIME str containing the the resource w accessed. The UTC format. T FILETIME me zero if the prov not support thi member.
RESOURCE_CREATIONTIME	adFileTime	Read-only. Ind FILETIME str containing the resource was c time is reporte Coordinated U Time (UTC) fo Read-only. Ind FILETIME str containing the the resource w accessed. The UTC format. T FILETIME me zero if the prov not support thi member.
RESOURCE_LASTACCESSTIME	AdFileTime	Read-only. Ind FILETIME str containing the the resource w accessed. The UTC format. T FILETIME me zero if the prov not support thi member.
RESOURCE_LASTWRITETIME	AdFileTime	Read-only. Ind FILETIME str containing the the resource w written. The ti UTC format. T FILETIME me zero if the prov not support thi member.

RESOURCE_STREAMSIZE	asUnsignedBigInt	Read-only. Indicates size of the resource default stream.
RESOURCE_ISCOLLECTION	AdBoolean	Read-only. True if resource is a collection such as a directory if the resource is a file.
RESOURCE_ISSTRUCTUREDDOCUMENT	AdBoolean	True if the resource is a structured document. False if the resource is not a structured document. It can be a collection or a file.
DEFAULT_DOCUMENT	AdVarWChar	Read-only. Indicates this resource's default URL to the default document of a structured document. Used when the stream is requested for a resource. This is blank for a structured document.
CHAPTERED_CHILDREN	AdChapter	Read-only. Operates on the children of the rowset container. (The <i>OLE DB for Internet Publishing</i> does not use this column.)
RESOURCE_DISPLAYNAME	AdVarWChar	Read-only. Indicates display name of resource.
RESOURCE_ISROOT	AdBoolean	Read-only. True if resource is the root of the collection or stream.

document.

See Also

[Record Object](#) | [Appendix A: Providers](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Section II: Remote Data Service (RDS)

This section contains the following chapters:

- [Chapter 11: RDS Fundamentals](#)
- [Chapter 12: RDS Tutorial](#)
- [Chapter 13: RDS Usage and Security](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Chapter 11: RDS Fundamentals

This section contains a series of topics that you can read in a prescribed order. The following topics are included:

- [Solutions for Remote Data Access](#)
- [Basic RDS Programming Model](#)
- [RDS Programming Model in Detail](#)
- [RDS Programming Model with Objects](#)
- [The RDS Object Model Summary](#)

This section also contains the [RDS tutorial](#) and an [RDS Scenario](#), which demonstrate how to access and update a data source, and a series of topics about [Using RDS](#), which discusses more advanced RDS topics.

See Also

[Solutions for Remote Data Access](#) | [RDS Tutorial](#) | [Using RDS](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Solutions for Remote Data Access

The Issue

ADO enables your application to directly gain access to and modify data sources (sometimes called a two-tier system). For example, if your connection is to the data source that contains your data, that is a direct connection in a two-tier system.

However, you may want to access data sources indirectly through an intermediary such as Microsoft® Internet Information Services (IIS). This arrangement is sometimes called a three-tier system. IIS is a client/server system that provides an efficient way for a local, or client, application to invoke a remote, or server, program across the Internet or an intranet. The server program gains access to the data source and optionally processes the acquired data.

For example, your intranet Web page contains an application written in Microsoft® Visual Basic Scripting Edition (VBScript), which connects to IIS. IIS in turn connects to the actual data source, retrieves the data, processes it in some way, and then returns the processed information to your application.

In this example, your application never directly connected to the data source; IIS did. And IIS accessed the data by means of ADO.

Note The client/server application does not have to be based on the Internet or an intranet (that is, Web-based)—it could consist solely of compiled programs on a local area network. However, the typical case is a Web-based application.

Because some visual control, such as a grid, check box, or list, may use the returned information, the returned information must be easily used by a visual control.

You want a simple and efficient application-programming interface that supports three-tier systems, and returns information as easily as if it had been retrieved on a two-tier system. Remote Data Service (RDS) is this interface.

The Solution

RDS defines a programming model—the sequence of activities necessary to gain access to and update a data source—to gain access to data through an intermediary, such as Internet Information Services (IIS). The programming model summarizes the entire functionality of RDS.

See Also

[Basic RDS Programming Model](#) | [RDS Tutorial](#) | [Using RDS](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Basic RDS Programming Model

RDS addresses applications that exist in the following environment: A client application specifies a program that will execute on a server and the parameters required to return the desired information. The program invoked on the server gains access to the specified data source, retrieves the information, optionally processes the data, and then returns the resulting information to your client application in a form that it can easily use. RDS provides the means for you to perform the following sequence of actions:

- Specify the program to be invoked on the server, and obtain a way to refer to it from the client. (This reference is sometimes called a *proxy*. It represents the remote server program. The client application will "call" the proxy as if it were a local program, but it actually invokes the remote server program.)
- Invoke the server program. Pass parameters to the server program that identify the data source and the command to issue. (The server program actually uses ADO to gain access to the data source. ADO makes a connection with one of the given parameters, and then issues the command specified in the other parameter.)
- The server program obtains a [Recordset](#) object from the data source. Optionally, the **Recordset** object is processed on the server.
- The server program returns the final **Recordset** object to the client application.
- On the client, the **Recordset** object is put into a form that can be easily used by visual controls.
- Any modifications to the **Recordset** object are sent back to the server program, which uses them to update the data source.

This programming model contains certain convenience features. If you do not need a complex server program to access the data source, and if you provide the required connection and command parameters, RDS will automatically retrieve the specified data with a simple, default server program.

If you need more complex processing, you can specify your own custom server program. For example, because a custom server program has the full power of ADO at its disposal, it could connect to several different data sources, combine

their data in some complex way, and then return a simple, processed result to the client application.

Finally, if your needs are somewhere in between, ADO now supports customizing the behavior of the default server program.

See Also

[RDS Programming Model in Detail](#) | [RDS Tutorial](#) | [Recordset Object](#) | [Using RDS](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

RDS Programming Model in Detail

The following are key elements of the RDS programming model:

- RDS.DataSpace
- RDSServer.DataFactory
- RDS.DataControl
- Event

RDS.DataSpace

Your client application must specify the server and the server program to invoke. In return, your application receives a reference to the server program and can treat the reference as if it were the server program itself.

The RDS object model embodies this functionality with the [RDS.DataSpace](#) object.

The server program is specified with a program identifier, or *ProgID*. The server uses the *ProgID* and the server machine's registry to locate information about the actual program to initiate.

RDS makes a distinction internally depending on whether the server program is on a remote server across the Internet or an intranet; a server on a local area network; or not on a server at all, but instead on a local dynamic-link library ([DLL](#)). This distinction determines how information is exchanged between the client and the server, and makes a tangible difference in the type of reference returned to your client application. However, from your point of view, this distinction has no special meaning. All that matters is that you receive a usable program reference.

RDSServer.DataFactory

RDS provides a default server program that can either perform a SQL query against the data source and return a [Recordset](#) object or take a **Recordset** object and update the data source.

The RDS object model embodies this functionality with the [RDSServer.DataFactory](#) object.

In addition, this object has a method for creating an empty **Recordset** object that you can fill programmatically ([CreateRecordset](#)), and another method for converting a **Recordset** object into a text string to build a Web page ([ConvertToString](#)).

With ADO, you can override some of the standard connection and command behavior of the **RDSServer.DataFactory** with a **DataFactory** handler and a customization file that contains connection, command, and security parameters.

The server program is sometimes called a *business object*. You can write your own custom business object that can perform complicated data access, validity checks, and so on. Even when writing a custom business object, you can create an instance of an **RDSServer.DataFactory** object and use some of its methods to accomplish your own tasks.

RDS.DataControl

RDS provides a means to combine the functionality of the **RDS.DataSpace** and **RDSServer.DataFactory**, and also enable visual controls to easily use the **Recordset** object returned by a query from a data source. RDS attempts, for the most common case, to do as much as possible to automatically gain access to information on a server and display it in a visual control.

The RDS object model embodies this functionality with the [RDS.DataControl](#) object.

The **RDS.DataControl** has two aspects. One aspect pertains to the data source. If you set the command and connection information using the **Connect** and **SQL** properties of the **RDS.DataControl**, it will automatically use the **RDS.DataSpace** to create a reference to the default **RDSServer.DataFactory** object. Then the **RDSServer.DataFactory** will use the **Connect** property value to connect to the data source, use the **SQL** property value to obtain a **Recordset** from the data source, and return the **Recordset** object to the **RDS.DataControl**.

The second aspect pertains to the display of returned **Recordset** information in a visual control. You can associate a visual control with the **RDS.DataControl** (in a process called binding) and gain access to the information in the associated **Recordset** object, displaying query results on a Web page in Microsoft® Internet Explorer. Each **RDS.DataControl** object binds one **Recordset** object, representing the results of a single query, to one or more visual controls (for example, a text box, combo box, grid control, and so forth). There may be more than one **RDS.DataControl** object on each page. Each **RDS.DataControl** object can be connected to a different data source and contain the results of a separate query.

The **RDS.DataControl** object also has its own methods for navigating, sorting, and filtering the rows of the associated **Recordset** object. These methods are similar, but not the same as the methods on the ADO **Recordset** object.

Events

RDS supports two of its own events, which are independent of the ADO event model. The [onReadyStateChange](#) event is called whenever the **RDS.DataControl ReadyState** property changes, thus notifying you when an asynchronous operation has successfully completed, terminated, or experienced an error. The [onError](#) event is called whenever an error occurs, even if the error occurs during an [asynchronous operation](#).

Note Microsoft Internet Explorer provides two additional events to RDS—**onDataSetChanged** (the **Recordset** is functional but still retrieving rows) and **onDataSetComplete** (the **Recordset** has finished retrieving rows).

See Also

[RDS Programming Model with Objects](#) | [DataControl Object \(RDS\)](#) | [DataFactory Object \(RDS Server\)](#) | [DataSpace Object \(RDS\)](#) | [RDS Tutorial](#) | [Using RDS](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

RDS Programming Model with Objects

The goal of RDS is to gain access to and update data sources through an intermediary such as IIS. The programming model specifies the sequence of activities necessary to accomplish this goal. The object model specifies the objects whose methods and properties affect the programming model.

RDS provides the means to perform the following sequence of actions:

- Specify the program to be invoked on the server, and obtain a way (proxy) to refer to it from the client ([RDS.DataSpace](#)).
- Invoke the server program. Pass parameters to the server program that identifies the data source and the command to issue (proxy or [RDS.DataControl](#)).
- The server program obtains a [Recordset](#) object from the data source, typically by using ADO. Optionally, the **Recordset** object is processed on the server ([RDSServer.DataFactory](#)).
- The server program returns the final **Recordset** object to the client application (proxy).
- On the client, the **Recordset** object is put into a form that can be easily used by visual controls (visual control and **RDS.DataControl**).
- Changes to the **Recordset** object are sent back to the server and used to update the data source (**RDS.DataControl** or **RDSServer.DataFactory**).

See Also

[RDS Object Model Summary](#) | [DataControl Object \(RDS\)](#) | [DataFactory Object \(RDSServer\)](#) | [DataSpace Object \(RDS\)](#) | [RDS Tutorial](#) | [Recordset Object](#) | [Using RDS](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

RDS Object Model Summary

Object	Description
RDS.DataSpace	This object contains a method to obtain a server proxy . The proxy may be the default or a custom server program (business object). The server program may be invoked on the Internet, an intranet, a local area network, or be a local dynamic-link library .
RDS.Server.DataFactory	This object represents the default server program. It executes the default RDS data retrieval and update behavior. This object can automatically invoke the RDS.DataSpace and RDS.Server.DataFactory objects.
RDS.DataControl	Use this object to invoke the default RDS data retrieval or update behavior. This object also provides the means for visual controls to access the returned Recordset object.

See Also

[RDS Fundamentals](#) | [RDS Tutorial](#) | [Using RDS](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

System Requirements for the Address Book Application

To set up the Address Book sample application, you need to meet the following software and database requirements:

Software Requirements

The server computer software requirements for running this Web application include:

- Microsoft Windows NT® Server 4.0, with Service Pack 3 or later, or Microsoft Windows® 2000 Server.
- Microsoft Internet Information Services (IIS) version 3.0 or later, with Active Server Pages.

The [client](#) computer software requirements for running this Web application include:

- Microsoft Internet Explorer 4.0 or later.
- Microsoft Windows NT 4.0 Workstation or Server, Windows 2000, or Microsoft Windows 98.

Database Requirements

To use this sample, you must have:

- An operational Microsoft® SQL Server version 6.5 or later database server.
- Privileges to create the database and populate it with the sample data.
- Verification of the populated data through Enterprise Manager or the ISQL utilities (called Query Analyzer in SQL Server 7.0).

If you do not have privileges, your database administrator may need to set up the system and give you access permission to the database, or set up the database for you.

See Also

[DataControl Object \(RDS\)](#) | [Running the Address Book Sample Application](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Running the Address Book Sample Application

To run the Address Book application, follow this procedure.

To run this application

1. Make sure that Microsoft SQL Server is running. Click **Start**, point to **Programs**, point to **Microsoft SQL Server 7.0**, and then click **Service Manager**. If there is a green arrow in the white circle, then SQL Server is running. If it is not (there will be a red square in the white circle), click **Start/Continue**.
2. In Microsoft Internet Explorer 4.0 or later, type the following address:

`http://webserver/RDS/AddressBook/AddrBook.asp`

where *webserver* is the name of the Web server where the RDS server components are installed.

3. You can then try various scenarios in the Address Book sample application, such as searching for a person based on his or her e-mail name, listing all people with the title "Program Manager," or editing existing records. Click **Find** to fill the data grid with all the available names.

See Also

[Address Book Data-Binding Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Address Book Data-Binding Object

The Address Book application uses the [RDS.DataControl](#) object to bind data from the SQL Server database to a visual object (in this case, a DHTML table) in the application's client HTML page. The event-driven VBScript program logic uses the [RDS.DataControl](#) to:

- Query the database, send updates to the database, and refresh the data grid.
- Allow users to move to the first, next, previous, or last record in the data grid.

The following code defines the **RDS.DataControl** component:

```
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"  
  ID=DC1 Width=1 Height=1>  
  <PARAM NAME="SERVER" VALUE="http://<%=Request.ServerVariables("SE  
  <PARAM NAME="CONNECT" VALUE="Provider=sqloledb;  
Initial Catalog=AddrBookDb;Integrated Security=SSPI;">  
</OBJECT>
```

The OBJECT tag defines the **RDS.DataControl** component in the program. The tag includes two types of parameters:

- Those associated with the generic OBJECT tag.
- Those specific to the **RDS.DataControl** object.

Generic OBJECT Tag Parameters

The following table describes the parameters associated with the OBJECT tag.

Parameter	Description
<i>CLASSID</i>	A unique, 128-bit number that identifies the type of embedded object to the system. This identifier is maintained in the local computer's system registry. (For the class IDs of the RDS.DataControl object, see RDS.DataControl Object .)
<i>ID</i>	Defines a document-wide identifier for the embedded object that is used to identify it in code.

RDS.DataControl Tag Parameters

The following table describes the parameters specific to the **RDS.DataControl** object. (For a complete list of the **RDS.DataControl** object parameters, and when to implement them, see [RDS.DataControl object](#).)

Parameter	Description
SERVER	If you are using HTTP, the value is the name of the server computer preceded by http://.
CONNECT	Provides the necessary connection information for the RDS.DataControl to connect to SQL Server.
SQL	Sets or returns the query string used to retrieve the Recordset .

See Also

[Address Book Command Buttons](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Address Book Command Buttons

The Address Book application includes the following command buttons:

- A [Find](#) button to submit a query to the database.
- A **Clear** button to clear the text boxes before starting a new search.
- An [Update Profile](#) button to save changes to an employee record.
- A [Cancel Changes](#) button to discard changes.

Find Button

Clicking the **Find** button activates the VBScript Find_OnClick Sub procedure, which builds and sends the SQL query. Clicking this button populates the data grid.

Building the SQL Query

The first part of the Find_OnClick Sub procedure builds the SQL query, one phrase at a time, by appending text strings to a global SQL SELECT statement. It begins by setting the variable myQuery to a SQL SELECT statement that requests all rows of data from the data source table. Next, the Sub procedure scans each of the four input boxes on the page.

Because the program uses the word like in building the SQL statements, the queries are substring searches rather than exact matches.

For example, if the **Last Name** box contained the entry "Berge" and the **Title** box contained the entry "Program Manager", the SQL statement (value of myQuery) would read:

```
Select FirstName, LastName, Title, Email, Building, Room, Phone from
```

If the query was successful, all persons with a last name containing the text "Berge" (such as Berge and Berger) and with a title containing the words "Program Manager" (for example, Program Manager, Advanced Technologies) are displayed in the HTML data grid.

Preparing and Sending the Query

The last part of the Find_OnClick Sub procedure consists of two statements. The first statement assigns the [SQL](#) property of the [RDS.DataControl](#) object equal to the dynamically built SQL query. The second statement causes the **RDS.DataControl** object (DC1) to query the database, and then display the new results of the query in the grid.

```
Sub Find_OnClick
    ' ...
    DC1.SQL = myQuery
    DC1.Refresh
End Sub
```

Update Profile Button

Clicking the **Update Profile** button activates the VBScript Update_OnClick Sub procedure, which executes the [RDS.DataControl](#) object's (DC1) [SubmitChanges](#) and [Refresh](#) methods.

```
Sub Update_OnClick
    DC1.SubmitChanges
    DC1.Refresh
End Sub
```

When DC1.SubmitChanges executes, the Remote Data Service packages all the update information and sends it to the server via HTTP. The update is all-or-nothing; if a part of the update is unsuccessful, none of the changes is made, and a status message is returned. DC1.Refresh isn't necessary after **SubmitChanges** with Remote Data Service, but it ensures fresh data.

Cancel Changes Button

Clicking **Cancel Changes** activates the VBScript Cancel_OnClick Sub procedure, which executes the [RDS.DataControl](#) object's (DC1) [CancelUpdate](#) method.

```
Sub Cancel_OnClick  
    DC1.CancelUpdate  
End Sub
```

When DC1.CancelUpdate executes, it discards any edits that a user has made to an employee record on the data grid since the last query or update. It restores the original values.

See Also

[Address Book Navigation Buttons](#) | [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Address Book Navigation Buttons

The Address Book application displays the navigation buttons at the bottom of the Web page. You can use the navigation buttons to navigate through the data in the HTML grid display by selecting either the first or last row of data, or rows adjacent to the current selection.

Navigation Sub Procedures

The Address Book application contains several procedures that allow users to click the **First**, **Next**, **Previous**, and **Last** buttons to move around the data.

For example, clicking the **First** button activates the VBScript `First_OnClick` Sub procedure. The procedure executes a [MoveFirst](#) method, which makes the first row of data the current selection. Clicking the **Last** button activates the `Last_OnClick` Sub procedure, which invokes the [MoveLast](#) method, making the last row of data the current selection. The remaining navigation buttons work in a similar fashion.

```
' Move to the first record in the bound Recordset.
Sub First_OnClick
    DC1.Recordset.MoveFirst
End Sub

' Move to the next record from the current position
' in the bound Recordset.
Sub Next_OnClick
    If Not DC1.Recordset.EOF Then      ' Cannot move beyond bottom record
        DC1.Recordset.MoveNext
        Exit Sub
    End If
End Sub

' Move to the previous record from the current position in the bound
' Recordset.
Sub Prev_OnClick
    If Not DC1.Recordset.BOF Then     ' Cannot move beyond top record.
        DC1.Recordset.MovePrevious
        Exit Sub
    End If
End Sub

' Move to the last record in the bound Recordset.
Sub Last_OnClick
    DC1.Recordset.MoveLast
End Sub
```

See Also

[DataControl Object \(RDS\)](#) | [MoveFirst](#), [MoveLast](#), [MoveNext](#), and

[MovePrevious Methods \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Chapter 12: RDS Tutorial

This tutorial illustrates using the RDS programming model to query and update a data source. First, it describes the steps necessary to accomplish this task. Then the tutorial is repeated in Microsoft® Visual Basic Scripting Edition and Microsoft® Visual J++®, featuring ADO for Windows Foundation Classes (ADO/WFC).

This tutorial is coded in different languages for two reasons:

- The documentation for RDS assumes the reader codes in Visual Basic. This makes the documentation convenient for Visual Basic programmers, but less useful for programmers who use other languages.
- If you are uncertain about a particular RDS feature and you know a little of another language, you might be able to resolve your question by looking for the same feature expressed in another language.

How the Tutorial is Presented

This tutorial is based on the RDS programming model. It discusses each step of the programming model individually. In addition, it illustrates each step with a fragment of Visual Basic code.

The code example is repeated in other languages with minimal discussion. Each step in a given programming language tutorial is marked with the corresponding step in the programming model and descriptive tutorial. Use the number of the step to refer to the discussion in the descriptive tutorial.

The RDS programming model is stated below. Use it as a roadmap as you proceed through the tutorial.

RDS Programming Model with Objects

- Specify the program to be invoked on the server, and obtain a way ([proxy](#)) to refer to it from the [client](#).
- Invoke the server program. Pass parameters to the server program that identifies the data source and the command to issue.
- The server program obtains a [Recordset](#) object from the data source, typically by using ADO. Optionally, the **Recordset** object is processed on the server.
- The server program returns the final **Recordset** object to the client application.
- On the client, the **Recordset** object is optionally put into a form that can be easily used by visual controls.
- Changes to the **Recordset** object are sent back to the server and used to update the data source.

See Also

[Step 1: Specify a Server Program \(RDS Tutorial\)](#) | [RDS Tutorial \(VBScript\)](#) | [RDS Tutorial \(Visual J++\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

RDS 2.5 

Step 2: Invoke the Server Program (RDS Tutorial)

When you invoke a method on the client *proxy*, the actual program on the server executes the method. In this step, you'll execute a query on the server.

Part A If you weren't using [RDS.Server.DataFactory](#) in this tutorial, the most convenient way to perform this step would be to use the [RDS.DataControl](#) object. The **RDS.DataControl** combines the previous step of creating a proxy, with this step, issuing the query.

Set the **RDS.DataControl** object [Server](#) property to identify where the server program should be instantiated; the [Connect](#) property to specify the connect string to access the data source; and the [SQL](#) property to specify the query command text. Then issue the [Refresh](#) method to cause the server program to connect to the data source, retrieve rows specified by the query, and return a **Recordset** object to the client.

This tutorial does not use the **RDS.DataControl**, but this is how it would look if it did:

```
Sub RDSTutorial12A()  
    Dim DC as New RDS.DataControl  
    DC.Server = "http://yourServer"  
    DC.Connect = "DSN=Pubs"  
    DC.SQL = "SELECT * FROM Authors"  
    DC.Refresh  
    ...
```

Nor does the tutorial invoke RDS with ADO objects, but this is how it would look if it did:

```
Dim rs as New ADODB.Recordset  
rs.Open "SELECT * FROM Authors", "Provider=MS Remote;Data Source=Pubs  
    "Remote Server=http://yourServer;Remote Provider=SQLOLEDB;"
```

Part B The general method of performing this step is to invoke the **RDS.Server.DataFactory** object [Query](#) method. That method takes a connect

string, which is used to connect to a data source, and a command text, which is used to specify the rows to be returned from the data source.

This tutorial uses the **DataFactory** object **Query** method:

```
Sub RDSTutorial12B()  
    Dim DS as New RDS.DataSpace  
    Dim DF  
    Dim RS as ADODB.Recordset  
    Set DF = DS.CreateObject("RDS.Server.DataFactory", "http://yourSer  
    Set RS = DF.Query ("DSN=Pubs", "SELECT * FROM Authors")  
    ...
```

See Also

[Step 3: Server Obtains a Recordset \(RDS Tutorial\)](#) | [RDS Tutorial \(VBScript\)](#) | [RDS Tutorial \(Visual J++\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Step 3: Server Obtains a Recordset (RDS Tutorial)

The server program uses the connect string and command text to query the data source for the desired rows. ADO is typically used to retrieve this **Recordset**, although other Microsoft data access interfaces, such as OLE DB, could be used.

A custom server program might look like this:

```
Public Function ServerProgram(cn as String, qry as String) as Object
Dim rs as New ADODB.Recordset
    rs.CursorLocation = adUseClient
    rs.Open qry, cn
    rs.ActiveConnection = Nothing
    Set ServerProgram = rs
End Function
```

See Also

[Step 4: Server Returns the Recordset \(RDS Tutorial\)](#) | [RDS Tutorial \(VBScript\)](#) | [RDS Tutorial \(Visual J++\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Step 4: Server Returns the Recordset (RDS Tutorial)

RDS converts the retrieved **Recordset** object to a form that can be sent back to the client (that is, it *marshals* the **Recordset**). The exact form of the conversion and how it is sent depends on whether the server is on the Internet or an intranet, a local area network, or is a dynamic-link library. However, this detail is not critical; all that matters is that RDS sends the **Recordset** back to the client.

On the client side, a **Recordset** object is returned and assigned to a local variable.

```
Sub RDSTutorial4()  
    Dim DS as New RDS.DataSpace  
    Dim RS as ADODB.Recordset  
    Dim DF as Object  
    Set DF = DS.CreateObject("RDSserver.DataFactory", "http://yourSer  
    Set RS = DF.Query("DSN=Pubs", "SELECT * FROM Authors")  
    ...
```

See Also

[Step 5: DataControl is Made Usable \(RDS Tutorial\)](#) | [RDS Tutorial \(VBScript\)](#) | [RDS Tutorial \(Visual J++\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Step 5: DataControl is Made Usable (RDS Tutorial)

The returned **Recordset** object is available for use. You can examine, navigate, or edit it as you would any other **Recordset**. What you can do with the **Recordset** depends on your environment. Visual Basic and Visual C++ have visual controls that can use a **Recordset** directly or indirectly with the aid of an enabling data control.

For example, if you are displaying a Web page in Microsoft Internet Explorer, you might want to display the **Recordset** object data in a visual control. Visual controls on a Web page cannot access a **Recordset** object directly. However, they can access the **Recordset** object through the [RDS.DataControl](#). The **RDS.DataControl** becomes usable by a visual control when its [SourceRecordset](#) property is set to the **Recordset** object.

The visual control object must have its **DATASRC** parameter set to the **RDS.DataControl**, and its **DATAFLD** property set to a **Recordset** object field (column).

In this tutorial, set the **SourceRecordset** property:

```
Sub RDSTutorial15()  
    Dim DS as New RDS.DataSpace  
    Dim RS as ADODB.Recordset  
    Dim DC as New RDS.DataControl  
    Dim DF as Object  
    Set DF = DS.CreateObject("RDSserver.DataFactory", "http://yourSer  
    Set RS = DF.Query ("DSN=Pubs", "SELECT * FROM Authors")  
    DC.SourceRecordset = RS           ' Visual controls can now bind to  
    ...
```

See Also

[Step 6: Changes are Sent to the Server \(RDS Tutorial\)](#) | [RDS Tutorial \(VBScript\)](#)
| [RDS Tutorial \(Visual J++\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 

Step 6: Changes are Sent to the Server (RDS Tutorial)

If the **Recordset** object is edited, any changes (that is, rows that are added, changed, or deleted) can be sent back to the server.

Note The default behavior of RDS can be invoked implicitly with ADO objects and the Microsoft OLE DB Remoting Provider. Queries can return **Recordsets**, and edited **Recordsets** can update the data source. This tutorial does not invoke RDS with ADO objects, but this is how it would look if it did:

```
Dim rs as New ADODB.Recordset
rs.Open "SELECT * FROM Authors", "Provider=MS Remote;Data Source=Pubs
      "Remote Server=http://yourServer;Remote Provider=SQLOLEDB;"
...      ' Edit the Recordset.
rs.UpdateBatch ' The equivalent of SubmitChanges.
...
```

Part A Assume for this case that you have only used the [RDS.DataControl](#) and that a **Recordset** object is now associated with the **RDS.DataControl**. The [SubmitChanges](#) method updates the data source with any changes to the **Recordset** object if the [Server](#) and [Connect](#) properties are still set.

```
Sub RDSTutorial16A()
Dim DC as New RDS.DataControl
Dim RS as ADODB.Recordset
DC.Server = "http://yourServer"
DC.Connect = "DSN=Pubs"
DC.SQL = "SELECT * FROM Authors"
DC.Refresh
...
Set RS = DC.Recordset
    ' Edit the Recordset.
...
DC.SubmitChanges
...
```

Part B Alternatively, you could update the server with the [RDS.Server.DataFactory](#) object, specifying a connection and a **Recordset** object.

```
Sub RDSTutorial6B()  
Dim DS As New RDS.DataSpace  
Dim RS As ADODB.Recordset  
Dim DC As New RDS.DataControl  
Dim DF As Object  
Dim blnStatus As Boolean  
Set DF = DS.CreateObject("RDS.Server.DataFactory", "http://yourServer  
Set RS = DF.Query ("DSN=Pubs", "SELECT * FROM Authors")  
DC.SourceRecordset = RS      ' Visual controls can now bind to DC.  
    ' Edit the Recordset.  
blnStatus = DF.SubmitChanges "DSN=Pubs", RS  
End Sub
```

This is the end of the tutorial.

See Also

[Microsoft OLE DB Remoting Provider](#) | [RDS Tutorial](#) | [RDS Tutorial \(VBScript\)](#) | [RDS Tutorial \(Visual J++\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

RDS Tutorial (VBScript)

This is the RDS Tutorial, written in Microsoft Visual Basic Scripting Edition. For a description of the purpose of this tutorial, see the [RDS Tutorial](#).

In this tutorial, [RDS.DataControl](#) and [RDS.DataSpace](#) are created at design time—that is, they are defined with object tags, like this: <OBJECT> . . . </OBJECT>. Alternatively, they could be created at run time with the **Server.CreateObject** method. For example, the **RDS.DataControl** object could be created like this:

```
Set DC = Server.CreateObject("RDS.DataControl")
  <!-- RDS.DataControl -->
  <OBJECT
    ID="DC1" CLASSID="CLSID:BD96C556-65A3-11D0-983A-00C04FC29E33">
  </OBJECT>

  <!-- RDS.DataSpace -->
  <OBJECT
    ID="DS1" WIDTH=1 HEIGHT=1
    CLASSID="CLSID:BD96C556-65A3-11D0-983A-00C04FC29E36">
  </OBJECT>

  <SCRIPT LANGUAGE="VBScript">

  Sub RDSTutorial()
  Dim DF1
```

Step 1—Specify a server program

VBScript can discover the name of the IIS Web server it is running on by accessing the VBScript **Request.ServerVariables** method available to Active Server Pages:

```
"http://<%=Request.ServerVariables("SERVER_NAME")%>"
```

However, for this tutorial, use the imaginary server, "yourServer".

Note Pay attention to the data type of **ByRef** arguments. VBScript does not let you specify the variable type, so you must always pass a Variant. When using HTTP, RDS will allow you to pass a Variant to a method that expects a non-Variant if you invoke it with the **RDS.DataSpace** object

[CreateObject](#) method. When using [DCOM](#) or an in-process server, match the parameter types on the client and server sides or you will receive a "Type Mismatch" error.

```
Set DF1 = DS1.CreateObject("RDS.Server.DataFactory", "http://yourServ
```

Step 2a—Invoke the server program with RDS.DataControl

This example is merely a comment demonstrating that the default behavior of the **RDS.DataControl** is to perform the specified query.

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID="DC1
  <PARAM NAME="SQL" VALUE="SELECT * FROM Authors">
  <PARAM NAME="Connect" VALUE="DSN=Pubs;">
  <PARAM NAME="Server" VALUE="http://yourServer/">
</OBJECT>
...
<SCRIPT LANGUAGE="VBScript">

Sub RDSTutorial2A()
  Dim RS
  DC1.Refresh
  Set RS = DC1.Recordset
  ...
```

Step 2b—Invoke the server program with RDS.Server.DataFactory

Step 3—Server obtains a Recordset

Step 4—Server returns the Recordset

```
Set RS = DF1.Query("DSN=Pubs;", "SELECT * FROM Authors")
```

Step 5—DataControl is made usable by visual controls

' Assign the returned recordset to the DataControl.

```
DC1.SourceRecordset = RS
```

Step 6a—Changes are sent to the server with RDS.DataControl

This example is merely a comment demonstrating how the **RDS.DataControl** performs updates.

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID="DC1"
  <PARAM NAME="SQL" VALUE="SELECT * FROM Authors">
  <PARAM NAME="Connect" VALUE="DSN=Pubs;">
  <PARAM NAME="Server" VALUE="http://yourServer/">
</OBJECT>
...
<SCRIPT LANGUAGE="VBScript">

Sub RDSTutorial6A()
Dim RS
DC1.Refresh
...
Set RS = DC1.Recordset
' Edit the Recordset object...
' The SERVER and CONNECT properties are already set from Step 2A.
Set DC1.SourceRecordset = RS
...
DC1.SubmitChanges
```

Step 6b—Changes are sent to the server with RDSServer.DataFactory

```
DF.SubmitChanges "DSN=Pubs", RS

End Sub
</SCRIPT>
</BODY>
</HTML>
```

This is the end of the tutorial.

See Also

[RDS Tutorial](#) | [RDS Tutorial \(Visual J++\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

RDS Tutorial (Visual J++)

ADO/WFC does not completely follow the RDS object model in that it does not implement the [RDS.DataControl](#) object. ADO/WFC only implements the client-side class, [RDS.DataSpace](#).

The **DataSpace** class implements one method, [CreateObject](#), which returns an [ObjectProxy](#) object. The **DataSpace** class also implements the [InternetTimeout](#) property.

The **ObjectProxy** class implements one method, [call](#), which can invoke any server-side [business object](#).

This is the beginning of the tutorial.

```
import com.ms.wfc.data.*;
public class RDSTutorial
{
    public void tutorial()
    {
// Step 1: Specify a server program.
        ObjectProxy obj =
            DataSpace.createObject(
                "RDSServer.DataFactory",
                "http://YourServer");

// Step 2: Server returns a Recordset.
        Recordset rs = (Recordset) obj.call(
            "Query",
            new Object[] {"DSN=Pubs;", "SELECT * FROM Authors"});

// Step 3: Changes are sent to the server.
        ... // Edit Recordset.
        obj.call(
            "SubmitChanges",
            new Object[] {"DSN=Pubs;", rs});
        return;
    }
}
```

This is the end of the tutorial.

See Also

[RDS Tutorial](#) | [RDS Tutorial \(VBScript\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Chapter 13: RDS Usage and Security

Use the information in this chapter to set up your server and use RDS quickly. This chapter includes specific configuration steps that you might need to take when implementing RDS, describes some of the key relationships between RDS and other technologies, and helps identify solutions to problems that you might encounter when setting up an RDS solution.

This section contains information about:

- [Configuring RDS](#)
- [Using Related Technologies with RDS](#)
- [DataFactory Customization](#)
- [Troubleshooting RDS](#)

See Also

[RDS Fundamentals](#) | [RDS Tutorial](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Configuring RDS

To implement RDS efficiently, be sure you are familiar with the various configurations available to you. This section includes important information about security and scalability in your implementation of RDS. See the following topics for information about configuring your computers to use RDS.

- [Granting Guest Privileges to a Web Server Computer](#)
- [Registering a Custom Business Object](#)
- [Marking Business Objects as Safe for Scripting](#)
- [Registering Business Objects on the Client for Use with DCOM](#)
- [Setting DCOM Stream Marshaling Format](#)
- [Enabling a DLL to Run on DCOM](#)
- [Configuring Virtual Servers on IIS](#)
- [Specifying Threads Per Processor on IIS](#)
- [Securing RDS Applications](#)
- [Configuring DataFactory for Safe or Unrestricted Modes](#)

See Also

[Using Related Technologies with RDS](#) | [DataFactory Customization](#) | [Troubleshooting RDS](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Granting Guest Privileges to a Web Server Computer

The anonymous Web server account (*IUSR_ComputerName*) must be added to the Guests local group on the Web server computer to use RDS.

To grant guest privileges to a Web server computer

1. On your Microsoft Windows® 2000 Server computer, click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Computer Management**.
2. In the console tree, in **Local Users and Groups**, click **Groups**.
3. Select the **Guests** local group. From the **Action** menu, choose **Properties**.
4. In the **Guests Properties** dialog box, click **Add**.
5. If the anonymous Web server account does not appear in the list in the **Select Users or Groups** dialog box, type its name (*IUSR_ComputerName*) into the bottom blank box, and then click **Add**.
6. Click **OK**.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Registering a Custom Business Object

To successfully launch a custom [business object](#) (.dll or .exe) through the Web server, the business object's ProgID must be entered into the registry as explained in this procedure. This RDS feature protects the security of your [Web server](#) by running only sanctioned executables.

Note For MDAC 2.0 and later, the default business object, [RDS.Server.DataFactory](#), is not registered by default during MDAC installation. However, if **RDS.Server.DataFactory** was registered as safe for execution on the computer prior to the installation, the registry entry is maintained for the new installation.

To register a custom business object

1. Click **Start** and then click **Run**.
2. Type **RegEdit** and click **OK**.
3. In the Registry Editor, navigate to the **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\W3SVC\ADCLaunch** registry key.
4. Select the **ADCLaunch** key, and then from the **Edit** menu, point to **New** and click **Key**.
5. Type the ProgID of your custom business object and click **Enter**. Leave the **Value** entry blank.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Marking Business Objects as Safe for Scripting

To help ensure a secure Internet environment, you need to mark any [business objects](#) instantiated with the [RDS.DataSpace](#) object's [CreateObject](#) method as "safe for scripting." You need to ensure they are marked as such in the License area of the system registry before they can be used in [DCOM](#).

To manually mark your business object as safe for scripting, create a text file with a .reg extension that contains the following text. The following two numbers enable the safe-for-scripting feature:

```
[HKEY_CLASSES_ROOT\CLSID\<MyActiveXGUID>\Implemented
Categories\{7DD95801-9882-11CF-9FA9-00AA006C42C4}]
[HKEY_CLASSES_ROOT\CLSID\<MyActiveXGUID>\Implemented
Categories\{7DD95802-9882-11CF-9FA9-00AA006C42C4}]
```

where *<MyActiveXGUID>* is the hexadecimal GUID number of your business object. Save it and merge it into your registry by using the Registry Editor or double-clicking the .reg file in Windows Explorer.

Business objects created in Microsoft® Visual Basic can be automatically marked as "safe for scripting" with the Package and Deployment Wizard. When the wizard asks you to specify safety settings, select **Safe for initialization** and **Safe for scripting**.

On the last step, the Application Setup Wizard creates an .htm and a .cab file. You can then copy these two files to the target computer and double-click the .htm file to load the page and correctly register the server.

Because the business object will be installed in the Windows\System32\Occache directory by default, move it to the Windows\System32 directory and change the **HKEY_CLASSES_ROOT\CLSID*<MyActiveXGUID>*\InprocServer32** registry key to match the correct path.

Note Business objects marked as safe for scripting or safe for initialization can be instantiated and initialized by anyone over the network. Any custom

business object must not be designed and implemented casually. It is imperative that such objects do not present a security threat that hackers can explore to gain access to the sensitive area of the hosting server and inflict damages.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Registering Business Objects on the Client for Use with DCOM

Custom [business objects](#) need to ensure that the [client side](#) can map their program name (ProgId) to an identifier ([CLSID](#)) that can be used over [DCOM](#). For this reason, the ProgID of the DCOM object must be in the client-side registry and map to the class ID of the server-side business object. For the other supported protocols (HTTP, HTTPS, and in-process), this is not necessary.

For example, if you expose a server-side business object called MyBObj with a specific class ID, for instance, "{00112233-4455-6677-8899-00AABBCCDDEE}", make sure that the following entries are added to the client-side registry:

```
[HKEY_CLASSES_ROOT]
\MyBObj
  \Clsid
  (Default) "{00112233-4455-6677-8899-00AABBCCDDEE}"
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Setting DCOM Stream Marshaling Format

A client computer using components from RDS 1.5 or earlier is not compatible with a server using components from RDS 2.0 or later. When using DCOM as the underlying protocol, the support for RDS 2.0 or later is more efficient in transporting [Recordset](#) objects. If your client is running components from RDS 1.5 or earlier, you can set your server to work with the previous RDS support (called RDS 1.0) or the newer RDS support (called RDS 2.0 or later). Set either of the following registry entries:

```
[HKEY_CLASSES_ROOT]
\CLSID
  \[58ECEE30-E715-11CF-B0E3-00AA003F000F}
    \ADTGOptions] "MarshalFormat"="RDS10"
```

-or-

```
[HKEY_CLASSES_ROOT]
\CLSID
  \[58ECEE30-E715-11CF-B0E3-00AA003F000F}
    \ADTGOptions] "MarshalFormat"="RDS20"
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Enabling a DLL to Run on DCOM

The following steps outline how to enable a [business object .dll](#) to use both [DCOM](#) and Microsoft® Internet Information Services (HTTP) via Component Services.

1. Create a new empty package in the Component Services MMC snap-in.

You will use the Component Services MMC snap-in to create a package and add the DLL into this package. This makes the .dll accessible through DCOM, but it removes the accessibility through IIS. (If you check in the registry for the .dll, the **Inproc** key is now empty; setting the Activation attribute, explained later in this topic, adds a value in the **Inproc** key.)

2. Install a business object into the package.

-or-

Import the [RDS Server Data Factory](#) object into the package.

3. Set the Activation attribute for the package to **In the creator's process** (Library application).

To make the .dll accessible through DCOM and IIS on the same computer, you must set the component's Activation attribute in the Component Services MMC snap-in. After you set the attribute to **In the creator's process**, you will notice that an **Inproc** server key in the registry has been added that points to a Component Services surrogate .dll.

For more information about Component Services (or Microsoft® Transaction Service, if you are using Windows NT) and how to perform these steps, see the [Component Services](#) documentation or visit the Transaction Server Web site at <http://www.microsoft.com/com/tech/mts.asp>.

RDS 2.5 

Configuring Virtual Servers on IIS

When creating virtual servers in Internet Information Services 4.0, the following two extra steps are needed in order to configure the virtual server to work with RDS:

1. When setting up the server, check "Allow Execute Access."
2. Move `msadcs.dll` to `vroot\msadc`, where `vroot` is the home directory of your virtual server.

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Specifying Threads Per Processor on IIS

When using RDS with Internet Information Services 4.0 or later, the number of threads created per processor can be controlled by manipulating the registry on the Web server. The number of threads per processor can affect performance in a high traffic situation, or in low traffic situations with large query sizes. The user should experiment for best results.

The method used to determine and change the default value for this setting depends upon the configuration of the IIS 4.0 server.

With MDAC 2.1.2.4202.3 (GA) or later installed on the IIS server, RDS uses the same Component Services (or Microsoft Transaction Services, if you are using Windows NT) thread pool as ASP scripts use. The default value for the number of threads per processor is 10. To change the default, you must add the following key to the server registry:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\InetInfo\Parame
```

where *MaxPoolThreads* is a REG_DWORD. *MaxPoolThreads* does not appear in the Registry unless it is specifically added. Valid values range from 5 to a recommended maximum of 20. If the value specified by the registry key is greater than the value of the *PoolThreadLimit* key (located under the same path), then *PoolThreadLimit* value is used.

Alternatively, if MDAC 2.1 2.1.1.3711.11 (GA) or earlier is installed on the IIS server, the default value for the number of threads per processor is 6. To change the default, you must add the following key to the registry on the IIS server:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameter
```

where *ADCThreads* is a REG_DWORD. *ADCThreads* does not appear in the Registry unless it is specifically added. The range of valid values is 1 to 50. If the value specified by the registry key is greater than 50, then the maximum value is used (50).

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Securing RDS Applications

Microsoft Internet Explorer Security Issues

With new security enhancements added to Microsoft® Internet Explorer, some ADO and RDS objects are restricted to running only in a "safe" mode environment. This requires that you are aware of these issues, including different zones, security levels, restrictive behavior, unsafe operations, and customized security settings.

For more information about these issues, see [ADO and RDS Security Issues in Microsoft Internet Explorer under ActiveX Data Objects \(ADO\) Technical Articles](#).

Security and Your Web Server

If you use the [RDSServer.DataFactory](#) object on your Internet [Web server](#), remember that doing so creates a potential security risk. External users who obtain valid data source name (DSN), user ID, and password information could write pages to send any query to that data source. If you want more restricted access to a data source, one option is to unregister and delete the **RDSServer.DataFactory** object (msadcf.dll), and instead use custom [business objects](#) with hard-coded queries.

For more information on the security implications of using the RDSServer.DataFactory object, see the Microsoft Security Bulletin MS99-025 on the Microsoft Security Web site at <http://www.microsoft.com/security/default.asp>.

Client Impersonation and Security

If the **Password Authentication** property for your IIS Web server is set to Windows NT Challenge/Response authentication (for Windows NT 4.0) or to Integrated Windows authentication (for Windows 2000), then business objects are invoked under the [client's](#) security context. This is a new feature in RDS 1.5 that allows Client Impersonation over HTTP. When working in this mode, the logon to the Web server (IIS) is not anonymous but uses the user ID and password that the client computer is running under. If the ODBC DSNs are set up to use Trusted Connection, then access to databases, such as SQL Server, also happens under the client's security context. But this only works if the database is on the same computer as the IIS; the client credentials cannot be carried over to yet another computer.

For example, a client, John Doe, with userid="JohnD" and password="secret" is logged on to a client computer. He runs a browser-based application that needs to access the **RDS**Server.DataFactory object to create a [Recordset](#) by executing an SQL query on the "MyServer" computer running IIS. MyServer, a system running Windows NT Server 4.0, is set up to use Windows NT Challenge/Response authentication, its [ODBC](#) DSN has "Use Trusted Connection" selected, and the server also contains the SQL Server data source. When a request is received on the Web server, it asks the [client](#) for the user ID and password. Thus, the request is logged on MyServer as coming from "JohnD"/"Secret" instead of IUSER_MyServer (which is the default when Anonymous Password Authentication is on). Similarly, when logging on to SQL Server, "JohnD"/"Secret" is used.

Consequently, the IIS Windows NT Challenge/Response authentication mode allows HTML pages to be created without the user being explicitly prompted for the user ID and password information needed to log on to the database. If the IIS Basic Authentication were being used, then this also would be required.

Password Authentication

RDS can communicate with an IIS Web server running in any one of the three Password Authentication modes: Anonymous, Basic, or NT Challenge/Response authentication (called Integrated Windows authentication in Windows 2000). These settings define how a Web server controls access through it, such as requiring that a client computer have explicit access privileges on the NT Web server.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Configuring DataFactory for Safe or Unrestricted Modes

By default, ADO is installed with a "safe" [RDSServer.DataFactory](#) configuration. Safe mode for RDS Server components means that the following are true:

1. Handler is required with the RDSServer.DataFactory (this is mandated by a registry key setting).
2. The default handler, msdfmap.handler, is registered, present in the safe-handler list, and marked as the default handler.
3. Msdfmap.ini file is installed in the Windows directory. You must configure this file according to your needs, before using RDS in three-tier mode.

Optionally, you can configure an unrestricted **DataFactory** installation. **DataFactory** can be used directly without the custom handler. Users can still use a custom handler by modifying the connection strings, but it is not required. For more information on the implications of using the **RDSServer.DataFactory** object, see [Securing RDS Applications](#).

The registry file handsafe.reg has been provided to set up the handler registry entries for a safe configuration. To run in safe mode, run handsafe.reg. The registry file handunsf.reg has been provided to set up the handler registry entries for an unrestricted configuration. To run in unrestricted mode, run handunsf.reg.

After running either handsafe.reg or handunsf.reg, you must stop and restart the World Wide Web Publishing Service on the Web server by typing the following commands in a command window: "NET STOP W3SVC" and "NET START W3SVC".

For more information about using the customization handler feature of RDS, see the technical article [Using the Customization Handler Feature in RDS 2.1](#).

See Also

[DataFactory Customization](#) | [RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Using Related Technologies with RDS

This section contains specific information about using RDS with aspects of the following technologies:

- [ODBC Connection Pooling](#)
- [Microsoft Component Services](#)

Many other technologies interact with Remote Data Service or are used in its implementation.

Internet Information Services For more information about Microsoft Internet Information Services (IIS) or Active Server Pages (ASP), see the IIS Web site at <http://www.microsoft.com/ntserver/web/exec/feature/Datasheet.asp>.

Microsoft Component Services For more information about Component Services (known as Microsoft Transaction Server in Windows NT), see the Windows Web page at <http://www.microsoft.com/windows/default.asp>. For information about Microsoft Transaction Server, see the MTS Web site at <http://www.microsoft.com/com/tech/mts.asp>.

Microsoft SQL Server For more information about Microsoft® SQL Server, see the SQL Server Web site at <http://www.microsoft.com/sql/>.

Microsoft Internet Explorer For more information about Microsoft Internet Explorer, see the Internet Explorer Web page at <http://www.microsoft.com/windows/ie/default.htm> and the MSDN Online Web Workshop page at <http://msdn.microsoft.com/workshop/default.asp>.

Microsoft Windows NT Server/Windows 2000 Server For more information about security in Microsoft Windows NT Server or Windows 2000 Server, see the Windows Web page at <http://www.microsoft.com/windows/default.asp>.

See Also

[RDS Fundamentals](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 

Using RDS with ODBC Connection Pooling

If you're using an [ODBC](#) data source, you can use the connection pooling option in Internet Information Services (IIS) to achieve high performance handling of client load. Connection pooling is a resource manager for connections, maintaining the open state on frequently used connections.

To enable connection pooling, refer to the Internet Information Services documentation.

Please note that enabling connection pooling may subject the [Web server](#) to other restrictions, as noted in the Microsoft Internet Information Services documentation.

To ensure that connection pooling is stable and provides additional performance gains, you must configure Microsoft SQL Server to use the TCP/IP Socket network library.

To do this, you need to:

- Configure the SQL Server computer to use TCP/IP Sockets.
- Configure the Web server to use TCP/IP Sockets.

Configuring the SQL Server Computer to Use TCP/IP Sockets

On the SQL Server computer, run the SQL Server Setup program so that interactions with the data source use the TCP/IP Socket network library.

To specify the TCP/IP Socket network library on the SQL Server computer

In Microsoft SQL Server 6.5:

1. From the **Start** menu, point to **Programs**, point to **Microsoft SQL Server 6.5**, and then click **SQL Setup**.
2. Click **Continue** twice.
3. In the **Microsoft SQL Server—Options** dialog box, select **Change Network Support**, and then click **Continue**.
4. Make sure the **TCP/IP Sockets** check box is selected, and click **OK**.
5. Click **Continue** to finish, and exit setup.

In Microsoft SQL Server 7.0:

1. From the **Start** menu, point to **Programs**, point to **Microsoft SQL Server 7.0**, and then click **Server Network Utility**.
2. On the **General** tab of the dialog box, click **Add**.
3. In the **Add Network Library Configuration** dialog box, click **TCP/IP**.
4. In the **Port number** and **Proxy address** boxes, enter the port number and proxy address provided by your network administrator.
5. Click **OK** to finish, and exit setup.

Configuring the Web Server to Use TCP/IP Sockets

There are two options for configuring the Web server to use TCP/IP Sockets. What you do depends on whether all SQL Servers are accessed from the Web server, or only a specific SQL Server is accessed from the [Web server](#).

If all SQL Servers are accessed from the Web server, you need to run the SQL Server Client Configuration Utility on the Web server computer. The following steps change the default network library for all SQL Server connections made from this IIS Web server to use the TCP/IP Sockets network library.

To configure the Web server (all SQL Servers)

For Microsoft SQL Server 6.5:

1. From the **Start** menu, point to **Programs**, point to **Microsoft SQL Server 6.5**, and then click **SQL Client Configuration Utility**.
2. Click the **Net Library** tab.
3. In the **Default Network** box, select **TCP/IP Sockets**.
4. Click **Done** to save changes and exit the utility.

For Microsoft SQL Server 7.0:

1. From the **Start** menu, point to **Programs**, point to **Microsoft SQL Server 7.0**, and then click **Client Network Utility**.
2. Click the **General** tab.
3. In the **Default network library** box, click **TCP/IP**.
4. Click **OK** to save changes and exit the utility.

If a specific SQL Server is accessed from a Web server, you need to run the SQL Server Client Configuration Utility on the Web server computer. To change the network library for a specific SQL Server connection, configure the SQL Server Client software on the Web server computer as follows.

To configure the Web server (a specific SQL Server)

For Microsoft SQL Server 6.5:

1. From the **Start** menu, point to **Programs**, point to **Microsoft SQL Server 6.5**, and then click **SQL Client Configuration Utility**.
2. Click the **Advanced** tab.
3. In the **Server** box, type the name of the server to connect to using **TCP/IP Sockets**.
4. In the **DLL Name** box, select **TCP/IP Sockets**.
5. Click **Add/Modify**. All data sources pointing to this server will now use TCP/IP Sockets.
6. Click **Done**.

For Microsoft SQL Server 7.0:

1. From the **Start** menu, point to **Programs**, point to **Microsoft SQL Server 7.0**, and then click **Client Configuration Utility**.
2. Click the **General** tab.
3. Click **Add**.
4. Enter the alias of the server in the **Server alias** box. In the **Network libraries** box, click **TCP/IP**. In the **Computer name** box, enter the computer name of the computer that listens for TCP/IP Sockets clients. In the **Port number** box, enter the port on which the SQL Server listens.
5. Click **OK**, and then **OK** again to exit the utility.

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Running Business Objects in Component Services

[Business objects](#) can be executable files (.exe) or [dynamic-link libraries](#) (.dll). The configuration you use to run the business object depends on whether the object is a .dll or .exe file:

- Business objects created as .exe files can be called through [DCOM](#). If these business objects are used through Internet Information Services (IIS), they are subject to additional [marshaling](#) of data, which will slow client performance.
- Business objects created as .dll files can be used via IIS (and therefore HTTP). They can also be used over [DCOM](#) only via Component Services (or Microsoft Transaction Server, if you are using Windows NT). Business object DLLs will need to be registered on the IIS server computer to give you accessibility via IIS. (For steps on how to configure a DLL to run on DCOM, see the section, "[Enabling a DLL to Run on DCOM](#).")

Note When business objects on the [middle tier](#) are implemented as Component Services components (using **GetObjectContext**, **SetComplete**, and **SetAbort**), they can use Component Services (or MTS, if you are using Windows NT) context objects to maintain their state across multiple client calls. This scenario is possible with DCOM, which is typically implemented between trusted clients and servers (an intranet). In this case, the [RDS.DataSpace](#) object and [CreateObject](#) method on the client side are replaced by the transaction context object and **CreateInstance** method (provided by the **ITransactionContext** interface), implemented by Component Services.

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

DataFactory Customization

Remote Data Service (RDS) provides a way to easily perform data access in a three-tier client/server system. A client data control specifies connection and command string parameters to perform a query on a remote data source, or connection string and [Recordset](#) object parameters to perform an update.

The parameters are passed to a server program, which performs the data-access operation on the remote data source. RDS provides a default server program called the [RDSServer.DataFactory](#) object. The **RDSServer.DataFactory** object returns any **Recordset** object produced by a query to the client.

However, the **RDSServer.DataFactory** is limited to performing queries and updates. It cannot perform any validation or processing on the connection or command strings.

With ADO, you can specify that the **DataFactory** work in conjunction with another type of server program called a *handler*. The handler can modify client connection and command strings before they are used to access the data source. In addition, the handler can enforce access rights, which govern the ability of the client to read and write data to the data source.

The parameters the handler uses to modify client parameters and access rights are specified in sections of a customization file.

See the following topics for more information about customizing the **DataFactory** object:

- [Understanding the Customization File](#)
- [Customization File Connect Section](#)
- [Customization File SQL Section](#)
- [Customization File UserList Section](#)
- [Customization File Logs Section](#)
- [Required Client Settings](#)
- [Writing Your Own Customized Handler](#)

RDS 2.5 

Understanding the Customization File

Each section header in the customization file consists of square brackets ([]) containing a type and parameter. The four section types are indicated by the literal strings **connect**, **sql**, **userlist**, or **logs**. The parameter is the literal string, the default, a user-specified identifier, or nothing.

Therefore, each section is marked with one of the following section headers:

```
[ connect    default    ]
[ connect    identifier ]
[ sql        default    ]
[ sql        identifier ]
[ userlist   identifier ]
[ logs                               ]
```

The section headers have the following parts.

Part	Description
connect	A literal string that modifies a connection string.
sql	A literal string that modifies a command string.
userlist	A literal string that modifies the access rights of a specific user.
logs	A literal string that specifies a log file recording operational errors.
default	A literal string that is used if no identifier is specified or found. A string that matches a string in the connect or command string.
<i>identifier</i>	<ul style="list-style-type: none">• Use this section if the section header contains connect and the identifier string is found in the connection string.• Use this section if the section header contains sql and the identifier string is found in the command

- string.
- Use this section if the section header contains **userlist** and the identifier string matches a **connect** section identifier.

The **DataFactory** calls the handler, passing client parameters. The handler searches for whole strings in the client parameters that match identifiers in the appropriate section headers. If a match is found, the contents of that section are applied to the client parameter.

A particular section is used under the following circumstances:

- A **connect** section is used if the value part of the client connect string keyword, "**Data Source=value**", matches a **connect** section identifier.
- An **sql** section is used if the client command string contains a string that matches an **sql** section identifier.
- A **connect** or **sql** section with a default parameter is used if there is no matching identifier.
- A **userlist** section is used if the **userlist** section identifier matches a **connect** section identifier. If there is a match, the contents of the **userlist** section are applied to the connection governed by the **connect** section.
- If the string in a connection or command string does not match the identifier in any **connect** or **sql** section header, and there is no **connect** or **sql** section header with a default parameter, then the client string is used without modification.
- The **logs** section is used whenever the **DataFactory** is in operation.

See Also

[Customization File Connect Section](#) | [Customization File Logs Section](#) | [Customization File SQL Section](#) | [Customization File UserList Section](#) | [DataFactory Customization](#) | [Required Client Settings](#) | [Writing Your Own Customized Handler](#)

RDS 2.5 

Customization File Connect Section

The default behavior of the handler is to deny all connections. The **connect** section specifies exceptions to that behavior. For example, if all the **connect** sections were absent or empty, then by default no connections could be made.

The **connect** section can contain:

- A default access entry that specifies the default read and write operations allowed on this connection. If there is no default access entry in the section, the section will be ignored.
- A new connection string that replaces the client connection string.

Syntax

A default access entry is of the form:

Access=accessRight

A replacement connection string entry is of the form:

Connect=connectionString

Part	Description
Connect	A literal string that indicates this is a connection string entry.
connectionString	A string that replaces the whole client connection string.
Access	A literal string that indicates this is an access entry. One of the following access rights:
accessRight	<ul style="list-style-type: none">• NoAccess — User cannot access the data source.• ReadOnly — User can read the data source.• ReadWrite — User can read or write to the data source.

If you want to allow any connection (in effect, disabling the default handler behavior), set the access entry in the **connect default** section to `Access=ReadWrite`, and delete or comment out any other **connect identifier** section.

See Also

[Customization File Logs Section](#) | [Customization File SQL Section](#) | [Customization File UserList Section](#) | [DataFactory Customization](#) | [Required Client Settings](#) | [Understanding the Customization File](#) | [Writing Your Own Customized Handler](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Customization File SQL Section

The **sql** section can contain a new SQL string that replaces the client command string. If there is no SQL string in the section, the section will be ignored.

The new SQL string may be *parameterized*. That is, parameters in the **sql** section SQL string (designated by the '?' character) can be replaced by corresponding arguments in an *identifier* in the client command string (designated by a comma-delimited list in parentheses). The identifier and argument list behave like a function call.

For example, assume the client command string is "CustomerByID(4)", the SQL section header is [SQL CustomerByID], and the new SQL section string is "SELECT * FROM Customers WHERE CustomerID = ?". The Handler will generate "SELECT * FROM Customers WHERE CustomerID = 4" and use that string to query the data source.

If the new SQL statement is the null string (""), then the section is ignored.

If the new SQL statement string is not valid, then the execution of the statement will fail. The client parameter is effectively ignored. You can do this intentionally to "turn off" all client SQL commands by specifying:

```
[SQL default]  
SQL = " "
```

Syntax

A replacement SQL string entry is of the form:

SQL=*sqlString*

Part	Description
SQL	A literal string that indicates this is an SQL section entry.
<i>sqlString</i>	An SQL string that replaces the client string.

See Also

[Customization File Connect Section](#) | [Customization File Logs Section](#) | [Customization File UserList Section](#) | [DataFactory Customization](#) | [Required Client Settings](#) | [Understanding the Customization File](#) | [Writing Your Own Customized Handler](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Customization File UserList Section

The **userlist** section pertains to the **connect** section with the same section *identifier* parameter.

This section can contain a *user access entry*, which specifies access rights for the specified user and overrides the *default access entry* in the matching **connect** section.

Syntax

A user access entry is of the form:

userName=***accessRights***

Part	Description
<i>userName</i>	The <i>user name</i> of the person employing this connection. Valid user names are established with the IIS Service Manager dialog.
<i>accessRights</i>	One of the following access rights: <ul style="list-style-type: none">• NoAccess — User cannot access the data source.• ReadOnly — User can read the data source.• ReadWrite — User can read or write to the data source.

See Also

[Customization File Connect Section](#) | [Customization File Logs Section](#) | [Customization File SQL Section](#) | [DataFactory Customization](#) | [Required Client Settings](#) | [Understanding the Customization File](#) | [Writing Your Own Customized Handler](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 

Customization File Logs Section

The **logs** section contains a log file entry, which specifies the name of a file that records errors during the operation of the **DataFactory**.

Syntax

A log file entry is of the form:

err=*FileName*

	Part	Description
err		A literal string that indicates this is a log file entry.
<i>FileName</i>		A complete path and file name. The typical file name is c:\msdfmap.log .

The log file will contain the user name, HRESULT, date, and time of each error.

See Also

[Customization File Connect Section](#) | [Customization File SQL Section](#) | [Customization File UserList Section](#) | [DataFactory Customization](#) | [Required Client Settings](#) | [Understanding the Customization File](#) | [Writing Your Own Customized Handler](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Required Client Settings

Specify the following settings to use a custom **DataFactory** handler.

- Specify "Provider=MS Remote" in the [Connection](#) object [Provider](#) property or the **Connection** object connection string "**Provider=**" keyword.
- Set the [CursorLocation](#) property to **adUseClient**.
- Specify the name of the handler to use in the [RDS.DataControl](#) object's **Handler** property, or the [Recordset](#) object's connection string "**Handler=**" keyword. (You cannot set the handler in the **Connection** object connect string.)

RDS provides a default handler on the server named **MSDFMAP.Handler**.
(The default customization file is named **MSDFMAP.INI**.)

Example

Assume that the following sections in **MSDFMAP.INI** and the data source name, AdvWorks, have been previously defined:

```
[connect CustomerDataBase]
Access=ReadWrite
Connect="DSN=AdvWorks"

[sql CustomerById]
SQL="SELECT * FROM Customers WHERE CustomerID = ?"
```

The following code snippets are written in Visual Basic:

RDS.DataControl Version

```
Dim dc as New RDS.DataControl
Set dc.Handler = "MSDFMAP.Handler"
Set dc.Server = "http://yourServer"
Set dc.Connect = "Data Source=CustomerDatabase"
Set dc.SQL = "CustomerById(4)"
dc.Refresh
```

Recordset Version

```
Dim rs as New ADODB.Recordset  
rs.CursorLocation = adUseClient
```

Specify either the [Handler](#) property or keyword; the **Provider** property or keyword; and the *CustomerById* and *CustomerDatabase* identifiers. Then open the **Recordset** object.

```
rs.Open "CustomerById(4)", "Handler=MSDFMAP.Handler;" & _  
    "Provider=MS Remote;Data Source=CustomerDatabase;" & _  
    "Remote Server=http://yourServer"
```

See Also

[Customization File Connect Section](#) | [Customization File Logs Section](#) |
[Customization File SQL Section](#) | [Customization File UserList Section](#) |
[DataFactory Customization](#) | [Microsoft OLE DB Remoting Provider](#) |
[Understanding the Customization File](#) | [Writing Your Own Customized Handler](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Writing Your Own Customized Handler

You may want to write your own handler if you are an IIS server administrator who wants the default RDS support, but more control over user requests and access rights.

The `MSDFMAP.Handler` implements the **IDataFactoryHandler** interface.

IDataFactoryHandler Interface

This interface has two methods, **GetRecordset** and **Reconnect**. Both methods require that the [CursorLocation](#) property be set to **adUseClient**.

Both methods take arguments that appear after the first comma in the "**Handler**=" keyword. For example, "Handler=progid, arg1, arg2;" will pass an argument string of "arg1, arg2", and "Handler=progid" will pass a null argument.

GetRecordset Method

This method queries the data source and creates a new [Recordset](#) object using the arguments provided. The **Recordset** must be opened with **adLockBatchOptimistic** and must not be opened asynchronously.

Arguments

conn The connection string.

args The arguments for the handler.

query The command text for making a query.

ppRS The pointer where the **Recordset** should be returned.

Reconnect Method

This method updates the data source. It creates a new [Connection](#) object and attaches the given **Recordset**.

Arguments

conn The connection string.

args The arguments for the handler.

pRS A **Recordset** object.

msdfhdl.idl

This is the interface definition for **IDataFactoryHandler** that appears in the **msdfhdl.idl** file.

```
[
  uuid(D80DE8B3-0001-11d1-91E6-00C04FBBBFB3),
  version(1.0)
]
library MSDFHDL
{
  importlib("stdole32.tlb");
  importlib("stdole2.tlb");

  // TLib : Microsoft ActiveX Data Objects 2.0 Library
  // {00000200-0000-0010-8000-00AA006D2EA4}
  #ifdef IMPLIB
  importlib("implib\\x86\\release\\ado\\msado15.dll");
  #else
  importlib("msado20.dll");
  #endif

  [
    odl,
    uuid(D80DE8B5-0001-11d1-91E6-00C04FBBBFB3),
    version(1.0)
  ]
  interface IDataFactoryHandler : IUnknown
  {
    HRESULT _stdcall GetRecordset(
      [in] BSTR conn,
      [in] BSTR args,
      [in] BSTR query,
      [out, retval] _Recordset **ppRS);

    // DataFactory will use the ActiveConnection property
    // on the Recordset after calling Reconnect.
    HRESULT _stdcall Reconnect(
      [in] BSTR conn,
      [in] BSTR args,
      [in] _Recordset *pRS);
  };
};
```

See Also

[Customization File Connect Section](#) | [Customization File Logs Section](#) |
[Customization File SQL Section](#) | [Customization File UserList Section](#) |
[DataFactory Customization](#) | [Required Client Settings](#) | [Understanding the
Customization File](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Troubleshooting RDS

Refer to this section for solutions to specific errors or issues with RDS. The issues discussed in this section are:

- [Configuring RDS on Windows 2000](#)
- ["Internet Server Error: Access Denied"](#)
- [RDS Returns "Stream Not Read" Error](#)
- [Deadlocks With Read Repeatable Isolation Level](#)
- [Ensuring Sufficient TempDB Space](#)
- [Minimizing Log File Space Usage](#)

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Configuring RDS on Windows 2000

If you experience difficulties getting RDS to function properly after upgrading to Windows 2000, follow the steps below to troubleshoot the issue.

1. Make sure that the World Wide Web Publishing Service is running first by navigating to `http://server` using Internet Explorer. If you are unable to access the web server this way, go to a command prompt and enter the following command, "NET START W3SVC".
2. From the Start menu, select Run. Type `msdfmap.ini` and click OK to open the `msdfmap.ini` file in Notepad. Check the [CONNECT DEFAULT] section, and if the ACCESS parameter is set to NOACCESS, change it to READONLY.
3. Using the RegEdit utility, navigate to "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DataFactory\Handler and make sure **HandlerRequired** is set to 0 and **DefaultHandler** is "" (Null string).

Note If you make any changes to this section of the registry, you must stop and restart the World Wide Web Publishing Service by entering the following commands at a command prompt: "NET STOP W3SVC" and "NET START W3SVC".

4. Using the RegEdit utility, navigate in the registry to "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SV and verify that there is a key called **RDS Server.Datafactory**. If not, create it.
5. Using Internet Services Manager, go to the Default Web Site and view the properties of the MSADC virtual root. Inspect the Directory Security/IP Address and Domain Name Restrictions. If the "Access is Denied" is checked then select "Granted".

Be sure to try rebooting the server if the changes do not appear to solve the problem at first.

See Also

RDS Fundamentals

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 

"Internet Server Error: Access Denied"

If you get this error, it usually means that Microsoft Internet Information Services (IIS) returned the following status:

HTTP_STATUS_DENIED 401

Make sure the directories accessed by IIS have proper permissions. RDS can communicate with an IIS Web server running in any one of the three Password Authentication modes: Anonymous, Basic, or NT Challenge/Response (called Integrated Windows authentication in Windows 2000). Also, the [Web server](#) must have permissions to the data source computer if it is a Windows NT/Windows 2000 computer.

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

RDS Returns "Stream Not Read" Error

"The Stream object could not be read because it is empty, or the current position is at the end of the Stream. For non-empty Streams, set the current position with the Position property. To determine if a Stream is empty, check the Size property."

If you get the error above, it may be a result of trying to use a parameterized hierarchical query over http. RDS does not permit you to remote parameterized hierarchies.

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Deadlocks With Read Repeatable Isolation Level

If a custom [business object](#) uses an isolation level of read repeatable to access a SQL Server, and the business object is called simultaneously by two [clients](#) that send a query and update in the same transaction, a deadlock is possible. Remote Data Service is designed to allow one of the processes to time out to release the deadlock, but the update will fail for that client.

Use the [Cursor Service Command Time Out](#) dynamic property to modify the length of the timeout.

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Ensuring Sufficient TempDB Space

If errors occur while handling [Recordset](#) objects that need processing space on Microsoft SQL Server 6.5, you may need to increase the size of the TempDB. (Some queries require temporary processing space; for example, a query with an ORDER BY clause requires a sort of the **Recordset**, which requires some temporary space.)

Important Read through this procedure before performing the actions because it isn't as easy to shrink a device once it is expanded.

Note By default in Microsoft SQL Server 7.0 and later, TempDB is set to automatically grow as needed. Therefore, this procedure may only be necessary on servers running versions earlier than 7.0.

To increase the TempDB space on SQL Server 6.5

1. Start Microsoft® SQL Server Enterprise Manager, open the tree for the Server, and then open the Database Devices tree.
2. Select a (physical) device to expand, such as Master, and double-click the device to open the **Edit Database Device** dialog box.

This dialog box shows how much space the current databases are using.

3. In the **Size** box, increase the device to the desired amount (for example, 50 megabytes (MB) of hard disk space).
4. Click **Change Now** to increase the amount of space to which the (logical) TempDB can expand.
5. Open the Databases tree on the server, and then double-click **TempDB** to open the **Edit Database** dialog box. The **Database** tab lists the amount of space currently allocated to TempDB (**Data Size**). By default, this is 2 MB.
6. Under the **Size** group, click **Expand**. The graphs show the available and allocated space on each of the physical devices. The bars in maroon color represent available space.
7. Select a **Log Device**, such as Master, to display the available size in the **Size (MB)** box.
8. Click **Expand Now** to allocate that space to the TempDB database.

The **Edit Database** dialog box displays the new allocated size for TempDB.

For more information about this topic, search the Microsoft SQL Server Enterprise Manager Help file for "Expand Database dialog box."

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 

Minimizing Log File Space Usage

A log file may fill quickly (thus halting the server) if there is a large volume of activity on an SQL Server database. You can set the log file to **Truncate on Checkpoint** to significantly extend the life of the log file for a database.

To enable Truncate on Checkpoint in Microsoft SQL Server 6.5

1. Start Microsoft SQL Server Enterprise Manager, open the tree for the Server, and then open the Database Devices tree.
2. Double-click the name of the database on which this feature will be enabled.
3. From the **Database** tab, select **Truncate**.
4. From the **Options** tab, select **Truncate Log on Checkpoint**, and then click **OK**.

To enable Truncate on Checkpoint in Microsoft SQL Server 7.0

1. Start Microsoft SQL Server Enterprise Manager, open the tree for the Server, and then open the Databases tree.
2. Right-click the name of the database on which this feature will be enabled and choose **Properties**.
3. From the **Options** tab, select **Truncate Log on Checkpoint**, and then click **OK**.

For more information about the **Truncate on Checkpoint** feature, see the Microsoft SQL Server documentation.

See Also

[RDS Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

Section III: ADO (Multidimensional) (ADO MD)

This section contains the following chapter:

- [Chapter 14: ADO MD Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

Chapter 14: ADO MD Fundamentals

Microsoft ActiveX Data Objects (Multidimensional) (ADO MD) provides easy access to multidimensional data from languages such as Microsoft Visual Basic, Microsoft Visual C++, and Microsoft Visual J++. ADO MD extends Microsoft ActiveX Data Objects (ADO) to include objects specific to multidimensional data, such as the [CubeDef](#) and [Cellset](#) objects. With ADO MD you can browse multidimensional schema, query a cube, and retrieve the results.

Like ADO, ADO MD uses an underlying OLE DB [provider](#) to gain access to data. To work with ADO MD, the provider must be a multidimensional data provider (MDP) as defined by the OLE DB for OLAP specification. MDPs present data in multidimensional views as opposed to tabular data providers (TDPs) that present data in tabular views. Refer to the documentation for your OLAP OLE DB provider for more detailed information on the specific syntax and behaviors supported by your provider.

This document assumes a working knowledge of the Visual Basic programming language and a general knowledge of ADO and OLAP. For more information, see the [ADO Programmer's Guide](#) and the OLE DB for OLAP Programmer's Reference. For more overview information about ADO MD, see the following topics:

- [Overview of Multidimensional Schemas and Data](#)
- [Working with Multidimensional Data](#)
- [Using ADO with ADO MD](#)
- [Programming with ADO MD](#)
- [ADO MD Object Model](#)

See Also

[ADO MD Object Model](#) | [Microsoft ADO Programmer's Guide](#) | [Microsoft ADOX Programmer's Reference](#) | [Overview of Multidimensional Schemas and Data](#) | [Programming with ADO MD](#) | [Using ADO with ADO MD](#) | [Working with Multidimensional Data](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 MD 

Overview of Multidimensional Schemas and Data

Understanding Multidimensional Schemas

The central metadata object in ADO MD is the *cube*, which consists of a structured set of related dimensions, hierarchies, levels, and members.

A *dimension* is an independent category of data from your multidimensional database, derived from your business entities. A dimension typically contains items to be used as query criteria for the measures of the database.

A *hierarchy* is a path of aggregation of a dimension. A dimension may have multiple levels of granularity, which have [parent-child relationships](#). A hierarchy defines how these levels are related.

A *level* is a step of aggregation in a hierarchy. For dimensions with multiple layers of information, each layer is a level.

A *member* is a data item in a dimension. Typically, you create a caption or describe a measure of the database using members.

Cubes are represented by [CubeDef](#) objects in ADO MD. Dimensions, hierarchies, levels, and members are also represented by their corresponding ADO MD objects: [Dimension](#), [Hierarchy](#), [Level](#), and [Member](#).

Dimensions

The dimensions of a cube depend on your business entities and types of data to be modeled in the database. Typically, each dimension is an independent entry point or mechanism for selecting data.

For example, a cube containing sales data has the following five dimensions: Salesperson, Geography, Time, Products, and Measures. The Measures dimension contains actual sales data values, while the other dimensions

represent ways to categorize and group the sales data values.

The Geography dimension has the following set of members:

{All, North America, Europe, Canada, USA, UK, Germany, Canada-West, Canada-East, USA-NW, USA-SW, USA-NE, USA-SE, England, Scotland, Wales, Ireland, Germany-North, Germany-South, Ottawa, Toronto, Vancouver, Calgary, Seattle, Boise, Los Angeles, Houston, Shreveport, Miami, Boston, New York, London, Dover, Glasgow, Edinburgh, Cardiff, Pembroke, Belfast, Berlin, Hamburg, Munich, Stuttgart}

Hierarchies

Hierarchies define the ways in which the levels of a dimension can be "rolled up" or grouped. A dimension can have more than one hierarchy. A natural hierarchy exists in the Geography dimension:



Levels

In the example Geography dimension pictured in the previous figure, each box represents a level in the hierarchy.

Each level has a set of members, as follows:

- The World = {All}
- Continents = {North America, Europe}
- Countries = {Canada, USA, UK, Germany}
- Regions = {Canada-East, Canada-West, USA-NE, USA-NW, USA-SE, USA-SW, England, Ireland, Scotland, Wales, Germany-North, Germany-South}
- Cities = {Ottawa, Toronto, Vancouver, Calgary, Seattle, Boise, Los Angeles, Houston, Shreveport, Miami, Boston, New York, London, Dover, Glasgow, Edinburgh, Cardiff, Pembroke, Belfast, Berlin, Hamburg, Munich, Stuttgart}

London, Dover, Glasgow, Edinburgh, Cardiff, Pembroke, Belfast, Berlin, Hamburg, Munich, Stuttgart}

Members

Members at the leaf level of a hierarchy have no [children](#), and members at the root level have no [parent](#). All other members have at least one parent and at least one child. For example, a partial traversal of the hierarchy tree in the Geography dimension yields the following parent-child relationships:

- {All} (parent of) {Europe, North America}
- {North America} (parent of) {Canada, USA}
- {USA} (parent of) {USA-NE, USA-NW, USA-SE, USA-SW}
- {USA-NW} (parent of) {Boise, Seattle}

Members can be consolidated along one or more hierarchies per dimension. Consider a Time dimension where there are two ways to roll up to the Year level from the Days level:



This example also illustrates another characteristic: Some members of the Week level of the Year-Week hierarchy do not appear in any level of the Year-Quarter hierarchy. Thus, a hierarchy need not include all members of a dimension.

See Also

[ADO MD Object Model](#) | [Microsoft ADO MD Programmer's Reference](#) | [Programming with ADO MD](#) | [Using ADO with ADO MD](#) | [Working with Multidimensional Data](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

Working with Multidimensional Data

A *cellset* is the result of a query on multidimensional data. It consists of a collection of axes, usually no more than four axes and typically only two or three. An *axis* is a collection of members from one or more dimensions, which is used to locate or filter specific values in a cube.

A *position* is a point along an axis. For an axis consisting of a single dimension, these positions are a subset of the dimension members. If an axis consists of more than one dimension, then each position is a compound entity, which has n parts where n is the number of dimensions oriented along that axis. Each part of the position is a member from one constituent dimension.

For example, if the Geography and Product dimensions from a cube containing sales data are oriented along the x-axis of a cellset, a position along this axis may contain the members "USA" and "Computers." In this example, determining a position along the x-axis requires that members from each dimension are oriented along the axis.

A *cell* is an object positioned at the intersection of axis coordinates. Each cell has multiple pieces of information associated with it, including the data itself, a formatted string (the displayable form of cell data), and the cell ordinal value. (Each cell is a unique ordinal value in the cellset. The ordinal value of the first cell in the cellset is zero, while the leftmost cell in the second row of a cellset with eight columns would have an ordinal value of eight.)

For example, a cube has the following six dimensions (note that this cube schema differs slightly from the example given in [Overview of Multidimensional Schemas and Data](#)):

- Salesperson
- Geography (natural hierarchy)—Continents, Countries, States, and so on
- Quarters—Quarters, Months, Days
- Years
- Measures—Sales, PercentChange, BudgetedSales
- Products

The following cellset represents sales for 1991 for all products:

Sales for 1991, All Products

		Valentine				Nash			
		USA			Japan	USA			Japan
		USA_North		USA_South		USA_North		USA_South	
		Seattle	Boston			Seattle	Boston		
Qtr 1	Jan	00	10	20	30	40	50	60	70
	Feb	01	11	21	31	41	51	61	71
	Mar	02	12	22	32	42	52	62	72
Qtr 2		03	13	23	33	43	53	63	73
Qtr 3		04	14	24	34	44	54	64	74
Qtr 4	Oct	05	15	25	35	45	55	65	75
	Nov	06	16	26	36	46	56	66	76
	Dec	07	17	27	37	47	57	67	77

Note The cell values in the example can be viewed as ordered pairs of axis position ordinals where the first digit represents the x-axis position and the second digit the y-axis position.

The characteristics of this cellset are as follows:

- Axis dimensions: Quarters, Salesperson, Geography
- Filter dimensions: Measures, Years, Products
- Two axes: COLUMN (x, or Axis 0) and ROW (y, or Axis 1)
- x-axis: two nested dimensions, Salesperson and Geography
- y-axis: Quarters dimension

The x-axis has two nested dimensions: Salesperson and Geography. From Geography, four members are selected: Seattle, Boston, USA-South, and Japan. Two members are selected from Salesperson: Valentine and Nash. This yields a total of eight positions on this axis (8 = 4*2).

Each coordinate is represented as a position with two members—one from the Salesperson dimension and another from the Geography dimension:

(Valentine, Seattle), (Valentine, Boston), (Valentine, USA_North), (Valentine, Japan), (Nash, Seattle), (Nash, Boston), (Nash, USA_Nort (Nash, Japan)

The y-axis has only one dimension, containing the following eight positions:

Jan, Feb, Mar, Qtr2, Qtr3, Oct, Nov, Dec

Cellsets, cells, axes, and positions are all represented in ADO MD by corresponding objects: [Cellset](#), [Cell](#), [Axis](#), and [Position](#).

See Also

[ADO MD Object Model](#) | [Microsoft ADO MD Programmer's Reference](#) | [Overview of Multidimensional Schemas and Data](#) | [Programming with ADO MD](#) | [Using ADO with ADO MD](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

Using ADO with ADO MD

ADO and ADO MD are related but separate object models. ADO provides objects for connecting to data sources, executing commands, retrieving tabular data and schema metadata in a tabular format, and viewing [provider](#) error information. ADO MD provides objects for retrieving multidimensional data and viewing multidimensional schema metadata.

When working with an MDP you may choose to use ADO, ADO MD, or both with your application. By referencing both libraries within your project, you will have full access to the functionality provided by your MDP.

It is often useful for consumers to get a flattened, tabular view of a multidimensional dataset. You can do this by using the ADO [Recordset](#) object. Specify the source for your [Cellset](#) as the **Source** parameter for the [Open](#) method of a **Recordset**, rather than as the source for an ADO MD **Cellset**.

It may also be useful to view the schema metadata in a tabular view rather than as a hierarchy of objects. The ADO [OpenSchema](#) method on the [Connection](#) object allows the user to open a **Recordset** containing schema information. The **QueryType** parameter of the **OpenSchema** method has several [SchemaEnum](#) values that relate specifically to MDPs. These values are:

- **adSchemaCubes**
- **adSchemaDimensions**
- **adSchemaHierarchies**
- **adSchemaLevels**
- **adSchemaMeasures**
- **adSchemaMembers**

To use ADO enum values with ADO MD properties or methods, your project must reference both the ADO and ADO MD libraries. For example, you can use the ADO **adState** enum values with the ADO MD [State](#) property. For more information about establishing references to libraries, see the documentation of your development tool.

For more information about the ADO objects and methods, see the [ADO API](#)

[Reference.](#)

See Also

[ADO MD Object Model](#) | [Microsoft ADO MD Programmer's Reference](#) | [Overview of Multidimensional Schemas and Data](#) | [Programming with ADO MD](#) | [Working with Multidimensional Data](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

Programming with ADO MD

To use ADO MD with your development tool, you should establish a reference to the ADO MD type library. The description of the ADO MD library is Microsoft ActiveX Data Objects (Multi-dimensional) Library. The ADO MD library filename is msadomd.dll, and the program ID (ProgID) is "ADOMD". For more information about establishing references to libraries, see the documentation of your development tool.

See Also

[Microsoft ADO MD Programmer's Reference](#) | [Overview of Multidimensional Schemas and Data](#) | [Using ADO with ADO MD](#) | [Working with Multidimensional Data](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Section IV: ADO Extensions for Data Definition Language and Security (ADOX)

This section contains the following chapter:

- [Chapter 15: ADOX Fundamentals](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Chapter 15: ADOX Fundamentals

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. Because it is an object-based approach to schema manipulation, you can write code that will work against various data sources regardless of differences in their native syntaxes.

ADOX is a companion library to the core ADO objects. It exposes additional objects for creating, modifying, and deleting schema objects, such as tables and procedures. It also includes security objects to maintain users and groups and to grant and revoke permissions on objects.

To use ADOX with your development tool, you should establish a reference to the ADOX type library. The description of the ADOX library is "Microsoft ADO Ext. for DDL and Security." The ADOX library file name is Msadox.dll, and the program ID (ProgID) is "ADOX". For more information about establishing references to libraries, see the documentation of your development tool.

The Microsoft OLE DB Provider for the Microsoft Jet Database Engine fully supports ADOX. Certain features of ADOX may not be supported, depending on your [data provider](#). For more information about supported features with the Microsoft OLE DB Provider for ODBC, the Microsoft OLE DB Provider for Oracle, or the Microsoft SQL Server OLE DB Provider, see the MDAC readme file.

This document assumes a working knowledge of the Microsoft® Visual Basic® programming language and a general knowledge of ADO. For more information about ADO, see the [ADO Programmer's Guide](#). For more overview information about ADOX, see the following topics:

- [ADOX Object Model](#)
- [ADOX Objects](#)
- [ADOX Collections](#)
- [ADOX Properties](#)
- [ADOX Methods](#)

- [ADOX Examples](#)

See Also

[ADOX API Reference](#) | [ADOX Code Examples](#) | [ADOX Collections](#) | [ADOX Enumerated Constants](#) | [ADOX Methods](#) | [ADOX Object Model](#) | [ADOX Objects](#) | [ADOX Properties](#) | [Microsoft ADO MD Programmer's Reference](#) | [ADO Programmer's Guide](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Provider Support for ADOX

Certain features of ADOX are unsupported, depending upon your OLE DB data provider. ADOX is fully supported with the [OLE DB Provider for Microsoft Jet](#). The unsupported features with the [Microsoft OLE DB Provider for SQL Server](#), the [Microsoft OLE DB Provider for ODBC](#), or the [Microsoft OLE DB Provider for Oracle](#) are listed below. ADOX is not supported by any other Microsoft OLE DB providers.

Microsoft OLE DB Provider for SQL Server

Object or Collection	Usage Restriction
Catalog object	The Create method is not supported.
Tables collection	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.

Microsoft OLE DB Provider for ODBC

Object or Collection	Usage Restriction
Catalog object	The Create method is not supported. The Append and Delete methods are not supported.
Tables collection	Properties are read/write prior to object creation, and read-only when referencing an existing object.
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.

Microsoft OLE DB Provider for Oracle

Object or Collection	Usage Restriction
Catalog object	The Create method is not supported. The Append and Delete methods are not supported.
Tables collection	Properties are read/write prior to object creation, and read-only when referencing an existing object.
Views collection	The Append and Delete methods are not supported.
View object	The Command property is not supported.
Procedures object	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.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Section V: Appendixes

The ADO Programmer's Guide contains the following appendixes:

- [Appendix A: Providers](#)
- [Appendix B: ADO Errors](#)
- [Appendix C: Programming with ADO](#)
- [Appendix D: ADO Samples](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Appendix A: Providers

This section addresses three kinds of providers: data providers, service providers, and service components. Providers fall into two categories: those providing data and those providing services. A *data provider* owns its own data and exposes it in tabular form to your application. A *service provider* encapsulates a service by producing and consuming data, augmenting features in your ADO applications. A service provider may also be further defined as a *service component*, which must work in conjunction with other service providers or components.

Data Providers

ADO is powerful and flexible because it can connect to any of several different data providers and still expose the same programming model, regardless of the specific features of any given provider.

However, because each data provider is unique, how your application interacts with ADO will vary slightly by data provider. The differences usually fall into one of three categories:

- Connection parameters in the [ConnectionString](#) property.
- [Command](#) object usage.
- Provider-specific [Recordset](#) behavior.

Details for each of the data providers currently available from Microsoft are listed as follows.

Area	Topic
ODBC databases	Microsoft OLE DB Provider for ODBC
Microsoft Indexing Service	Microsoft OLE DB Provider for Microsoft Indexing Service
Microsoft Active Directory Service	Microsoft OLE DB Provider for Microsoft Active Directory Service
Microsoft Jet databases	OLE DB Provider for Microsoft Jet
Microsoft SQL Server	Microsoft OLE DB Provider for SQL Server
Oracle databases	Microsoft OLE DB Provider for Oracle
Internet Publishing	Microsoft OLE DB Provider for Internet Publishing

Provider-Specific Dynamic Properties

The [Properties](#) collections of the [Connection](#), [Command](#), and [Recordset](#) objects include [dynamic properties](#) specific to the provider. These properties provide information about functionality specific to the provider beyond the built-in properties that ADO supports.

After establishing the connection and creating these objects, use the [Refresh](#) method on the object's **Properties** collection to obtain the provider-specific properties. Refer to the provider documentation and the OLE DB Programmer's Reference for detailed information about these dynamic properties.

Service Providers

To use a service provider, you must supply a keyword. You should also be aware of the provider-specific dynamic properties associated with each service provider. Provider-specific details are listed for each of the service providers currently available from Microsoft:

- [Microsoft Data Shaping Service for OLE DB](#)
- [Microsoft OLE DB Persistence Provider](#)
- [Microsoft OLE DB Remoting Provider](#)

Service Components

The [Cursor Service for OLE DB](#) service component supplements the [cursor](#) support functions of [data providers](#). It also requires a keyword and has dynamic properties.

For more information about providers, see the documentation for Microsoft OLE DB in the Microsoft Data Access Components SDK or visit the [Microsoft Universal Data Access Web site](#).

Provider Commands

For each provider listed here, if your applications allow users to enter SQL statements as the provider commands, you must always validate the user input and be vigilant of possible hacker attacks using potentially dangerous SQL statement, such as, `DROP TABLE t1`, as part of the user input.

See Also

[Command Object](#) | [Connection Object](#) | [Microsoft OLE DB Provider for Internet Publishing](#) | [Microsoft OLE DB Provider for Microsoft Active Directory Service](#) | [Microsoft OLE DB Provider for Microsoft Indexing Service](#) | [Microsoft OLE DB Provider for ODBC](#) | [Microsoft OLE DB Provider for Oracle](#) | [Microsoft OLE DB Provider for SQL Server](#) | [OLE DB Provider for Microsoft Jet](#) | [Properties Collection](#) | [Recordset Object](#) | [Refresh Method \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Microsoft OLE DB Provider for ODBC

To an ADO or RDS programmer, an ideal world would be one in which every data source exposes an OLE DB interface, so that ADO could call directly into the data source. Although increasingly more database vendors are implementing OLE DB interfaces, some data sources are not yet exposed this way. However, virtually all DBMS systems in use today can be accessed through [ODBC](#).

ODBC drivers are available for every major DBMS in use today, including Microsoft SQL Server, Microsoft Access (Microsoft Jet database engine), and Microsoft FoxPro, in addition to non-Microsoft database products such as Oracle.

The Microsoft ODBC Provider, however, allows ADO to connect to any ODBC data source. The [provider](#) is free-threaded and Unicode enabled.

The provider supports transactions, although different DBMS engines offer different types of transaction support. For example, Microsoft Access supports nested transactions up to five levels deep.

This is the default provider for ADO, and all provider-dependent ADO properties and methods are supported.

Connection String Parameters

To connect to this provider, set the **Provider=** argument of the [ConnectionString](#) property to:

MSDASQL

Reading the [Provider](#) property will return this string as well.

Typical Connection String

A typical connection string for this provider is:

```
"Provider=MSDASQL;DSN=dsnName;UID=userName;PWD=userPassword;"
```

The string consists of these keywords:

Keyword	Description
Provider	Specifies the OLE DB Provider for ODBC.
DSN	Specifies the data source name.
UID	Specifies the user name.
PWD	Specifies the user password.
URL	Specifies the URL of a file or directory published in a Web folder.

Because this is the default provider for ADO, if you omit the **Provider=** parameter from the connection string, ADO will attempt to establish a connection to this provider.

The provider does not support any specific connection parameters in addition to those defined by ADO. However, the provider will pass any non-ADO connection parameters to the ODBC driver manager.

Because you can omit the **Provider** parameter, you can therefore compose an ADO connection string that is identical to an ODBC connection string for the same data source. Use the same parameter names (**DRIVER=**, **DATABASE=**, **DSN=**, and so on), values, and syntax as you would when composing an ODBC connection string. You can connect with or without a predefined data source name (DSN) or FileDSN.

Syntax with a DSN or FileDSN:

```
"[Provider=MSDASQL;] { DSN=name | FileDSN=filename } ;  
[DATABASE=database;] UID=user; PWD=password"
```

Syntax without a DSN (DSN-less connection):

```
"[Provider=MSDASQL;] DRIVER=driver; SERVER=server;  
DATABASE=database; UID=user; PWD=password"
```

If you use a **DSN** or **FileDSN**, it must be defined through the ODBC Data Source Administrator in the Windows Control Panel. In Microsoft Windows 2000, the ODBC Administrator is located under Administrative Tools. In previous versions of Windows, the ODBC Administrator icon is named **32-bit ODBC** or simply **ODBC**.

As an alternative to setting a **DSN**, you can specify the ODBC driver (**DRIVER=**), such as "SQL Server;" the server name (**SERVER=**); and the database name (**DATABASE=**).

You can also specify a user account name (**UID=**), and the password for the user account (**PWD=**) in the ODBC-specific parameters or in the standard ADO-defined *user* and *password* parameters.

Although a **DSN** definition already specifies a database, you can specify a *database* parameter in addition to a **DSN** to connect to a different database. It is a good idea to always include *the database* parameter when you use a **DSN**. This will ensure that you connect to the proper database in the event that another user changed the default database parameter since you last checked the **DSN** definition.

Provider-Specific Connection Properties

The OLE DB provider for ODBC adds several properties to the [Properties](#) collection of the **Connection** object. The following table lists these properties with the corresponding OLE DB property name in parentheses.

Property Name	Description
Accessible Procedures (KAGPROP_ACCESSIBLEPROCEDURES)	Indicates whether the user has access to stored procedures.
Accessible Tables (KAGPROP_ACCESSIBLETABLES)	Indicates whether the user has permission to execute SELECT statements against the database tables.
Active Statements (KAGPROP_ACTIVESTATEMENTS)	Indicates the number of handles an ODBC driver can support on a connection.
Driver Name (KAGPROP_DRIVERNAME)	Indicates the file name of the ODBC driver.
Driver ODBC Version (KAGPROP_DRIVERODBCVER)	Indicates the version of ODBC that this driver supports.
File Usage (KAGPROP_FILEUSAGE)	Indicates how the driver treats a file in a data source; as a table or as a catalog.
Like Escape Clause (KAGPROP_LIKEESCAPECLAUSE)	Indicates whether the driver supports the definition and use of an escape character for the percent character (%) and underline character (_) in the LIKE predicate of a WHERE clause.
Max Columns in Group By (KAGPROP_MAXCOLUMNSINGROUPBY)	Indicates the maximum number of columns that can be listed in the GROUP BY clause of a SELECT statement.
Max Columns in Index	Indicates the maximum number of columns that can be included

(KAGPROP_MAXCOLUMNSININDEX)	in an index.
Max Columns in Order By (KAGPROP_MAXCOLUMNSINORDERBY)	Indicates the maximum number of columns that can be listed in the ORDER BY clause of a SELECT statement.
Max Columns in Select (KAGPROP_MAXCOLUMNSINSELECT)	Indicates the maximum number of columns that can be listed in the SELECT portion of a SELECT statement.
Max Columns in Table (KAGPROP_MAXCOLUMNSINTABLE)	Indicates the maximum number of columns allowed in a table.
Numeric Functions (KAGPROP_NUMERICFUNCTIONS)	Indicates which numeric functions are supported by the ODBC driver. For a listing of function names and the associated values used in this bitmask , see Appendix E: Scalar Functions in the ODBC documentation.
Outer Join Capabilities (KAGPROP_OJCAPABILITY)	Indicates the types of OUTER JOINS supported by the provider.
Outer Joins (KAGPROP_OUTERJOINS)	Indicates whether the provider supports OUTER JOINS.
Special Characters (KAGPROP_SPECIALCHARACTERS)	Indicates which characters have special meaning for the ODBC driver.
Stored Procedures (KAGPROP_PROCEDURES)	Indicates whether stored procedures are available for use with this ODBC driver.
String Functions (KAGPROP_STRINGFUNCTIONS)	Indicates which string functions are supported by the ODBC driver. For a listing of function names and the associated values used in this bitmask , see Appendix E: Scalar Functions in the ODBC documentation.

System Functions
(KAGPROP_SYSTEMFUNCTIONS)

Indicates which system functions are supported by the ODBC driver. For a listing of function names and the associated values used in this [bitmask](#), see Appendix E: Scalar Functions in the ODBC documentation.

Time/Date Functions
(KAGPROP_TIMEDATEFUNCTIONS)

Indicates which time and date functions are supported by the ODBC driver. For a listing of function names and the associated values used in this [bitmask](#), see Appendix E: Scalar Functions in the ODBC documentation.

SQL Grammar Support
(KAGPROP_ODBCSQLCONFORMANCE)

Indicates the SQL grammar that the ODBC driver supports.

Provider-Specific Recordset and Command Properties

The OLE DB provider for ODBC adds several properties to the **Properties** collection of the **Recordset** and **Command** objects. The following table lists these properties with the corresponding OLE DB property name in parentheses.

Property Name	Description
Query Based Updates/Deletes/Inserts (KAGPROP_QUERYBASEDUPDATES)	Indicates whether updates, deletions, and insertions can be performed using SQL queries.
ODBC Concurrency Type (KAGPROP_CONCURRENCY)	Indicates the method used to reduce potential problems caused by two users attempting to access the same data from the data source simultaneously.
BLOB accessibility on Forward-Only cursor (KAGPROP_BLOBSONFOCURSOR)	Indicates whether BLOB Fields can be accessed when using a forward-only cursor.
Include SQL_FLOAT, SQL_DOUBLE, and SQL_REAL in QBU WHERE clauses (KAGPROP_INCLUDENONEXACT)	Indicates whether SQL_FLOAT, SQL_DOUBLE, and SQL_REAL values can be included in a QBU WHERE clause.
Position on the last row after insert (KAGPROP_POSITIONONNEWROW)	Indicates that after a new record has been inserted in a table, the last row in the table will be come the current row.
IRowsetChangeExtInfo (KAGPROP_IROWSETCHANGEEXTINFO)	Indicates whether the IRowsetChange interface provides extended information support.
ODBC Cursor Type (KAGPROP_CURSOR)	Indicates the type of cursor used by the Recordset .
Generate a Rowset that can be marshaled (KAGPROP_MARSHALLABLE)	Indicates that the ODBC driver generates a recordset that can be marshaled

Command Text

How you use the [Command](#) object largely depends on the data source, and what type of query or command statement it will accept.

ODBC provides a specific syntax for calling stored procedures. For the [CommandText](#) property of a **Command** object, the *CommandText* argument to the **Execute** method on a [Connection](#) object, or the *Source* argument to the **Open** method on a [Recordset](#) object, passes in a string with this syntax:

```
"{ [ ? = ] call procedure [ ( ? [, ? [ , ] ] ) ] }"
```

Each ? references an object in the [Parameters](#) collection. The first ? references **Parameters(0)**, the next ? references **Parameters(1)**, and so on.

The parameter references are optional and depend on the structure of the stored procedure. If you want to call a stored procedure that defines no parameters, your string would look like this:

```
"{ call procedure }"
```

If you have two query parameters, your string would look like this:

```
"{ call procedure ( ?, ? ) }"
```

If the stored procedure will return a value, the return value is treated as another parameter. If you have no query parameters but you do have a return value, your string would look like this:

```
"{ ? = call procedure }"
```

Finally, if you have a return value and two query parameters, your string would look like this:

```
"{ ? = call procedure ( ?, ? ) }"
```

Recordset Behavior

The following tables list the standard ADO methods and properties available on a **Recordset** object opened with this provider.

For more detailed information about **Recordset** behavior for your provider configuration, run the [Supports](#) method and enumerate the **Properties** collection of the **Recordset** to determine whether provider-specific dynamic properties are present.

Availability of standard ADO **Recordset** properties:

Property	ForwardOnly	Dynamic	Keyset	Static
AbsolutePage	not available	not available	read/write	read/write
AbsolutePosition	not available	not available	read/write	read/write
ActiveConnection	read/write	read/write	read/write	read/write
BOF	read-only	read-only	read-only	read-only
Bookmark	not available	not available	read/write	read/write
CacheSize	read/write	read/write	read/write	read/write
CursorLocation	read/write	read/write	read/write	read/write
CursorType	read/write	read/write	read/write	read/write
EditMode	read-only	read-only	read-only	read-only
Filter	read/write	read/write	read/write	read/write
LockType	read/write	read/write	read/write	read/write
MarshalOptions	read/write	read/write	read/write	read/write
MaxRecords	read/write	read/write	read/write	read/write
PageCount	read/write	not available	read-only	read-only
PageSize	read/write	read/write	read/write	read/write
RecordCount	read/write	not available	read-only	read-only
Source	read/write	read/write	read/write	read/write
State	read-only	read-only	read-only	read-only
Status	read-only	read-only	read-only	read-only

The [AbsolutePosition](#) and [AbsolutePage](#) properties are write-only when ADO is

used with version 1.0 of the Microsoft OLE DB Provider for ODBC.

Availability of standard ADO **Recordset** methods:

Method	ForwardOnly	Dynamic	Keyset	Static
AddNew	Yes	Yes	Yes	Yes
Cancel	Yes	Yes	Yes	Yes
CancelBatch	Yes	Yes	Yes	Yes
CancelUpdate	Yes	Yes	Yes	Yes
Clone	No	No	Yes	Yes
Close	Yes	Yes	Yes	Yes
Delete	Yes	Yes	Yes	Yes
GetRows	Yes	Yes	Yes	Yes
Move	Yes	Yes	Yes	Yes
MoveFirst	Yes	Yes	Yes	Yes
MoveLast	No	Yes	Yes	Yes
MoveNext	Yes	Yes	Yes	Yes
MovePrevious	No	Yes	Yes	Yes
NextRecordset*	Yes	Yes	Yes	Yes
Open	Yes	Yes	Yes	Yes
Requery	Yes	Yes	Yes	Yes
Resync	No	No	Yes	Yes
Supports	Yes	Yes	Yes	Yes
Update	Yes	Yes	Yes	Yes
UpdateBatch	Yes	Yes	Yes	Yes

*Not supported for Microsoft Access databases.

Dynamic Properties

The Microsoft OLE DB Provider for ODBC inserts several dynamic properties into the **Properties** collection of the unopened [Connection](#), [Recordset](#), and [Command](#) objects.

The tables below are a cross-index of the ADO and OLE DB names for each dynamic property. The OLE DB Programmer's Reference refers to an ADO property name by the term, "Description." You can find more information about these properties in the OLE DB Programmer's Reference. Search for the OLE DB property name in the Index or see Appendix C: OLE DB Properties.

Connection Dynamic Properties

The following properties are added to the **Connection** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Active Sessions	DBPROP_ACTIVESESSIONS
Asynchable Abort	DBPROP_ASYNCTXNABORT
Asynchable Commit	DBPROP_ASYNCTNXCOMMIT
Autocommit Isolation Levels	DBPROP_SESS_AUTOCOMMITISOLEVELS
Catalog Location	DBPROP_CATALOGLOCATION
Catalog Term	DBPROP_CATALOGTERM
Column Definition	DBPROP_COLUMNDEFINITION
Connect Timeout	DBPROP_INIT_TIMEOUT
Current Catalog	DBPROP_CURRENTCATALOG
Data Source	DBPROP_INIT_DATASOURCE
Data Source Name	DBPROP_DATASOURCENAME
Data Source Object Threading Model	DBPROP_DSOTHREADMODEL
DBMS Name	DBPROP_DBMSNAME
DBMS Version	DBPROP_DBMSVER
Extended Properties	DBPROP_INIT_PROVIDERSTRING
GROUP BY Support	DBPROP_GROUPBY
Heterogeneous Table Support	DBPROP_HETEROGENEOUSTABLES
Identifier Case Sensitivity	DBPROP_IDENTIFIER_CASE
Initial Catalog	DBPROP_INIT_CATALOG
Isolation Levels	DBPROP_SUPPORTEDTXNISOLEVELS
Isolation Retention	DBPROP_SUPPORTEDTXNISORETAIN
Locale Identifier	DBPROP_INIT_LCID
Location	DBPROP_INIT_LOCATION

Maximum Index Size	DBPROP_MAXINDEXSIZE
Maximum Row Size	DBPROP_MAXROWSIZE
Maximum Row Size Includes BLOB	DBPROP_MAXROWSIZEINCLUDESBLOB
Maximum Tables in SELECT	DBPROP_MAXTABLESINSELECT
Mode	DBPROP_INIT_MODE
Multiple Parameter Sets	DBPROP_MULTIPLEPARAMSETS
Multiple Results	DBPROP_MULTIPLERESULTS
Multiple Storage Objects	DBPROP_MULTIPLESTORAGEOBJECTS
Multi-Table Update	DBPROP_MULTITABLEUPDATE
NULL Collation Order	DBPROP_NULLCOLLATION
NULL Concatenation Behavior	DBPROP_CONCATNULLBEHAVIOR
OLE DB Services	DBPROP_INIT_OLEDBSERVICES
OLE DB Version	DBPROP_PROVIDEROLEDBVER
OLE Object Support	DBPROP_OLEOBJECTS
Open Rowset Support	DBPROP_OPENROWSET SUPPORT
ORDER BY Columns in Select List	DBPROP_ORDERBYCOLUMNSINSELECT
Output Parameter Availability	DBPROP_OUTPUTPARAMETERAVAILABILITY
Password	DBPROP_AUTH_PASSWORD
Pass By Ref Accessors	DBPROP_BYREFACCESSORS
Persist Security Info	DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO
Persistent ID Type	DBPROP_PERSISTENTIDTYPE
Prepare Abort Behavior	DBPROP_PREPAREABORTBEHAVIOR
Prepare Commit Behavior	DBPROP_PREPARECOMMITBEHAVIOR
Procedure Term	DBPROP_PROCEDURETERM

Prompt	DBPROP_INIT_PROMPT
Provider Friendly Name	DBPROP_PROVIDERFRIENDLYNAME
Provider Name	DBPROP_PROVIDERFILENAME
Provider Version	DBPROP_PROVIDERVER
Read-Only Data Source	DBPROP_DATASOURCEREADONLY
Rowset Conversions on Command	DBPROP_ROWSETCONVERSIONSONCOMMAND
Schema Term	DBPROP_SCHEMATERM
Schema Usage	DBPROP_SCHEMAUSAGE
SQL Support	DBPROP_SQLSUPPORT
Structured Storage	DBPROP_STRUCTUREDSTORAGE
Subquery Support	DBPROP_SUBQUERIES
Table Term	DBPROP_TABLETERM
Transaction DDL	DBPROP_SUPPORTEDTXNDDL
User ID	DBPROP_AUTH_USERID
User Name	DBPROP_USERNAME
Window Handle	DBPROP_INIT_HWND

Recordset Dynamic Properties

The following properties are added to the **Recordset** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Access Order	DBPROP_ACCESSORDER
Blocking Storage Objects	DBPROP_BLOCKINGSTORAGEOBJECTS
Bookmark Type	DBPROP_BOOKMARKTYPE
Bookmarkable	DBPROP_IROWSETLOCATE
Change Inserted Rows	DBPROP_CHANGEINSERTEDROWS
Column Privileges	DBPROP_COLUMNRESTRICT
Column Set Notification	DBPROP_NOTIFYCOLUMNSET
Delay Storage Object Updates	DBPROP_DELAYSTORAGEOBJECTS
Fetch Backwards	DBPROP_CANFETCHBACKWARDS
Hold Rows	DBPROP_CANHOLDROWS
IAccessor	DBPROP_IAccessor
IColumnsInfo	DBPROP_IColumnsInfo
IColumnsRowset	DBPROP_IColumnsRowset
IConnectionPointContainer	DBPROP_IConnectionPointContainer
IConvertType	DBPROP_IConvertType
Immobile Rows	DBPROP_IMMOBILEROWS
IRowset	DBPROP_IRowset
IRowsetChange	DBPROP_IRowsetChange
IRowsetIdentity	DBPROP_IRowsetIdentity
IRowsetInfo	DBPROP_IRowsetInfo
IRowsetLocate	DBPROP_IRowsetLocate
IRowsetResynch	
IRowsetUpdate	DBPROP_IRowsetUpdate
ISequentialStream	DBPROP_ISequentialStream
ISupportErrorInfo	DBPROP_ISupportErrorInfo
Literal Bookmarks	DBPROP_LITERALBOOKMARKS

Literal Row Identity	DBPROP_LITERALIDENTITY
Maximum Open Rows	DBPROP_MAXOPENROWS
Maximum Pending Rows	DBPROP_MAXPENDINGROWS
Maximum Rows	DBPROP_MAXROWS
Notification Granularity	DBPROP_NOTIFICATIONGRANULARITY
Notification Phases	DBPROP_NOTIFICATIONPHASES
Objects Transacted	DBPROP_TRANSACTEDOBJECT
Own Changes Visible	DBPROP_OWNUPDATEDELETE
Own Inserts Visible	DBPROP_OWNINSERT
Preserve on Abort	DBPROP_ABORTPRESERVE
Preserve on Commit	DBPROP_COMMITPRESERVE
Quick Restart	DBPROP_QUICKRESTART
Reentrant Events	DBPROP_REENTRANTEVENTS
Remove Deleted Rows	DBPROP_REMOVEDELETED
Report Multiple Changes	DBPROP_REPORTMULTIPLECHANGES
Return Pending Inserts	DBPROP_RETURNPENDINGINSERTS
Row Delete Notification	DBPROP_NOTIFYROWDELETE
Row First Change Notification	DBPROP_NOTIFYROWFIRSTCHANGE
Row Insert Notification	DBPROP_NOTIFYROWINSERT
Row Privileges	DBPROP_ROWRESTRICT
Row Resynchronization Notification	DBPROP_NOTIFYROWRESYNCH
Row Threading Model	DBPROP_ROWTHREADMODEL
Row Undo Change Notification	DBPROP_NOTIFYROWUNDOCHANGE
Row Undo Delete Notification	DBPROP_NOTIFYROWUNDODELETE
Row Undo Insert Notification	DBPROP_NOTIFYROWUNDOINSERT
Row Update Notification	DBPROP_NOTIFYROWUPDATE
Rowset Fetch Position Change Notification	DBPROP_NOTIFYROWSETFETCHPOSITIONCH
Rowset Release	

Notification	DBPROP_NOTIFYROWSETRELEASE
Scroll Backwards	DBPROP_CANSCROLLBACKWARDS
Skip Deleted Bookmarks	DBPROP_BOOKMARKSKIPPED
Strong Row Identity	DBPROP_STRONGIDENTITY
Unique Rows	DBPROP_UNIQUEROWS
Updatability	DBPROP_UPDATABILITY
Use Bookmarks	DBPROP_BOOKMARKS

Command Dynamic Properties

The following properties are added to the **Command** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Access Order	DBPROP_ACCESSORDER
Blocking Storage Objects	DBPROP_BLOCKINGSTORAGEOBJECTS
Bookmark Type	DBPROP_BOOKMARKTYPE
Bookmarkable	DBPROP_IROWSETLOCATE
Change Inserted Rows	DBPROP_CHANGEINSERTEDROWS
Column Privileges	DBPROP_COLUMNRESTRICT
Column Set Notification	DBPROP_NOTIFYCOLUMNSET
Delay Storage Object Updates	DBPROP_DELAYSTORAGEOBJECTS
Fetch Backwards	DBPROP_CANFETCHBACKWARDS
Hold Rows	DBPROP_CANHOLDROWS
IAccessor	DBPROP_IAccessor
IColumnsInfo	DBPROP_IColumnsInfo
IColumnsRowset	DBPROP_IColumnsRowset
IConnectionPointContainer	DBPROP_IConnectionPointContainer
IConvertType	DBPROP_IConvertType
Immobile Rows	DBPROP_IMMOBILEROWS
IRowset	DBPROP_IRowset
IRowsetChange	DBPROP_IRowsetChange
IRowsetIdentity	DBPROP_IRowsetIdentity
IRowsetInfo	DBPROP_IRowsetInfo
IRowsetLocate	DBPROP_IRowsetLocate
IRowsetResynch	
IRowsetUpdate	DBPROP_IRowsetUpdate
ISequentialStream	DBPROP_ISequentialStream
ISupportErrorInfo	DBPROP_ISupportErrorInfo
Literal Bookmarks	DBPROP_LITERALBOOKMARKS

Literal Row Identity	DBPROP_LITERALIDENTITY
Maximum Open Rows	DBPROP_MAXOPENROWS
Maximum Pending Rows	DBPROP_MAXPENDINGROWS
Maximum Rows	DBPROP_MAXROWS
Notification Granularity	DBPROP_NOTIFICATIONGRANULARITY
Notification Phases	DBPROP_NOTIFICATIONPHASES
Objects Transacted	DBPROP_TRANSACTEDOBJECT
Own Changes Visible	DBPROP_OWNUPDATEDELETE
Own Inserts Visible	DBPROP_OWNINSERT
Preserve on Abort	DBPROP_ABORTPRESERVE
Preserve on Commit	DBPROP_COMMITPRESERVE
Quick Restart	DBPROP_QUICKRESTART
Reentrant Events	DBPROP_REENTRANTEVENTS
Remove Deleted Rows	DBPROP_REMOVEDELETED
Report Multiple Changes	DBPROP_REPORTMULTIPLECHANGES
Return Pending Inserts	DBPROP_RETURNPENDINGINSERTS
Row Delete Notification	DBPROP_NOTIFYROWDELETE
Row First Change Notification	DBPROP_NOTIFYROWFIRSTCHANGE
Row Insert Notification	DBPROP_NOTIFYROWINSERT
Row Privileges	DBPROP_ROWRESTRICT
Row Resynchronization Notification	DBPROP_NOTIFYROWRESYNCH
Row Threading Model	DBPROP_ROWTHREADMODEL
Row Undo Change Notification	DBPROP_NOTIFYROWUNDOCHANGE
Row Undo Delete Notification	DBPROP_NOTIFYROWUNDODELETE
Row Undo Insert Notification	DBPROP_NOTIFYROWUNDOINSERT
Row Update Notification	DBPROP_NOTIFYROWUPDATE
Rowset Fetch Position Change Notification	DBPROP_NOTIFYROWSETFETCHPOSITIONCH
Rowset Release	

Notification	DBPROP_NOTIFYROWSETRELEASE
Scroll Backwards	DBPROP_CANSCROLLBACKWARDS
Skip Deleted Bookmarks	DBPROP_BOOKMARKSKIP
Strong Row Identity	DBPROP_STRONGIDENTITY
Updatability	DBPROP_UPDATABILITY
Use Bookmarks	DBPROP_BOOKMARKS

See Also For details regarding specific implementation and functional information about the Microsoft OLE DB Provider for ODBC, consult the documentation for OLE DB Provider for ODBC and the Microsoft OLE DB Programmer's Reference available in the MDAC SDK or visit the [Universal Data Access Web site](#).

See Also

[Command Object](#) | [CommandText Property](#) | [Connection Object](#) | [ConnectionString Property](#) | [Execute Method \(ADO Command\)](#) | [Open Method \(ADO Recordset\)](#) | [Parameters Collection](#) | [Properties Collection](#) | [Provider Property](#) | [Recordset Object](#) | [Supports Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Microsoft OLE DB Provider for Microsoft Indexing Service

The Microsoft OLE DB Provider for Microsoft Indexing Service provides programmatic read-only access to file system and Web data indexed by Microsoft Indexing Service. ADO applications can issue SQL queries to retrieve content and file property information.

The [provider](#) is free-threaded and unicode enabled.

Connection String Parameters

To connect to this provider, set the **Provider=** argument to the [ConnectionString](#) property to:

MSIDXS

Reading the [Provider](#) property will return this string as well.

Typical Connection String

A typical connection string for this provider is:

```
"Provider=MSIDX;Data Source=myCatalog;Locale Identifier=nnnn;"
```

The string consists of these keywords:

Keyword	Description
Provider	Specifies the OLE DB Provider for Microsoft Indexing Service. Typically this is the only keyword specified in the connection string.
Data Source	Specifies the Indexing Service catalog name. If this keyword is not specified, the default system catalog is used.
Locale Identifier	Specifies a unique 32-bit number (for example, 1033) that specifies preferences related to the user's language. These preferences indicate how dates and times are formatted, items are sorted alphabetically, strings are compared, and so on. If this keyword is not specified, the default system locale identifier is used.

Command Text

The Indexing Service SQL query syntax consists of extensions to the SQL-92 **SELECT** statement and its **FROM** and **WHERE** clauses. The results of the query are returned via OLE DB [rowsets](#), which can be consumed by ADO and manipulated as [Recordset](#) objects.

You can search for exact words or phrases, or use wildcards to search for patterns or stems of words. The search logic can be based on Boolean decisions, weighted terms, or proximity to other words. You can also search by "free text," which finds matches based on meaning, rather than exact words.

The specific command dialect is fully documented in the [Microsoft Indexing Service Reference](#).

The provider does not accept stored procedure calls or simple table names (for example, the [CommandType](#) property will always be **adCmdText**).

Recordset Behavior

The following tables list the features available with a **Recordset** object opened with this provider. Only the Static [cursor](#) type (**adOpenStatic**) is available.

For more detailed information about **Recordset** behavior for your provider configuration, run the [Supports](#) method and enumerate the [Properties](#) collection of the **Recordset** to determine whether provider-specific dynamic properties are present.

Availability of standard ADO **Recordset** properties:

Property	Availability
AbsolutePage	read/write
AbsolutePosition	read/write
ActiveConnection	read-only
BOF	read-only
Bookmark *	read/write
CacheSize	read/write
CursorLocation	always adUseServer
CursorType	always adOpenStatic
EditMode	always adEditNone
EOF	read-only
Filter	read/write
LockType	read/write
MarshalOptions	not available
MaxRecords	read/write
PageCount	read-only
PageSize	read/write
RecordCount	read-only
Source	read/write
State	read-only
Status	read-only

*Bookmarks must be enabled on the provider in order for this feature to exist on the **Recordset**.

Availability of standard ADO **Recordset** methods:

Method	Available?
AddNew	No
Cancel	Yes
CancelBatch	No
CancelUpdate	No
Clone	Yes
Close	Yes
Delete	No
GetRows	Yes
Move	Yes
MoveFirst	Yes
NextRecordset	Yes
Open	Yes
Requery	Yes
Resync	Yes
Supports	Yes
Update	No
UpdateBatch	No

See Also For specific implementation details and functional information about the Microsoft OLE DB Provider for Microsoft Indexing Service, consult the Microsoft OLE DB Programmer's Reference and the [Microsoft Indexing Service](#) documentation, or visit the [Microsoft Internet Information Services Web page](#).

See Also

[CommandType Property](#) | [ConnectionString Property](#) | [Properties Collection](#) | [Provider Property](#) | [Recordset Object](#) | [Supports Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Microsoft OLE DB Provider for Microsoft Active Directory Service

The Microsoft Active Directory Service Interfaces (ADSI) Provider allows ADO to connect to heterogeneous directory services through ADSI. This gives ADO applications read-only access to the Microsoft Windows NT 4.0 and Microsoft Windows 2000 directory services, in addition to any LDAP-compliant directory service and Novell Directory Services. ADSI itself is based on a provider model, so if there is a new provider giving access to another directory, the ADO application will be able to access it seamlessly. The ADSI provider is free-threaded and unicode enabled.

Connection String Parameters

To connect to this provider, set the **Provider** argument of the [ConnectionString](#) property to:

ADSDS00bject

Reading the [Provider](#) property will return this string as well.

Typical Connection String

A typical connection string for this provider is:

```
"Provider=ADSDS00object;User ID=userName;Password=userPassword;"
```

The string consists of these keywords:

Keyword	Description
Provider	Specifies the OLE DB Provider for Microsoft Active Directory Service.
User ID	Specifies the user name. If this keyword is omitted, then the current logon is used.
Password	Specifies the user password. If this keyword is omitted, then the current logon is used.

Command Text

A four-part command text string is recognized by the [provider](#) in the following syntax:

```
"Root; Filter; Attributes[; Scope]"
```

Value	Description
<i>Root</i>	Indicates the ADsPath object from which to start the search (that is, the root of the search).
<i>Filter</i>	Indicates the search filter in the RFC 1960 format.
<i>Attributes</i>	Indicates a comma-delimited list of attributes to be returned.
	Optional. A String that specifies the scope of the search. Can be one of the following:
<i>Scope</i>	<ul style="list-style-type: none">• Base — Search only the base object (root of the search).• OneLevel — Search only one level.• Subtree — Search the entire subtree.

For example:

```
"<LDAP://DC=ArcadiaBay,DC=COM>;(objectClass=*);sn, givenName; subtre
```

The provider also supports SQL SELECT for command text. For example:

```
"SELECT title, telephoneNumber From 'LDAP://DC=Microsoft, DC=COM' WH  
objectClass='user' AND objectCategory='Person'"
```

The provider does not accept stored procedure calls or simple table names (for example, the [CommandType](#) property will always be **adCmdText**). See the Active Directory Service Interfaces documentation for a more complete description of the command text elements.

Recordset Behavior

The following tables list the features available on a [Recordset](#) object opened with this provider. Only the Static [cursor](#) type (**adOpenStatic**) is available.

For more detailed information about **Recordset** behavior for your provider configuration, run the [Supports](#) method and enumerate the [Properties](#) collection of the **Recordset** to determine whether provider-specific [dynamic properties](#) are present.

Availability of standard ADO **Recordset** properties:

Property	Availability
AbsolutePage	read/write
AbsolutePosition	read/write
ActiveConnection	read-only
BOF	read-only
Bookmark	read/write
CacheSize	read/write
CursorLocation	always adUseServer
CursorType	always adOpenStatic
EditMode	always adEditNone
EOF	read-only
Filter	read/write
LockType	read/write
MarshalOptions	not available
MaxRecords	read/write
PageCount	read-only
PageSize	read/write
RecordCount	read-only
Source	read/write
State	read-only
Status	read-only

Availability of standard ADO **Recordset** methods:

Method	Available?
AddNew	No
Cancel	No
CancelBatch	No
CancelUpdate	No
Clone	Yes
Close	Yes
Delete	No
GetRows	Yes
Move	Yes
MoveFirst	Yes
MoveLast	Yes
MoveNext	Yes
MovePrevious	Yes
NextRecordset	Yes
Open	Yes
Requery	Yes
Resync	Yes
Supports	Yes
Update	No
UpdateBatch	No

See Also For more general information about ADSI and the specifics of the provider, please refer to the Active Directory Service Interfaces documentation or visit the [ADSI Web page](#).

See Also

[CommandType Property](#) | [ConnectionString Property](#) | [Properties Collection](#) | [Provider Property](#) | [Recordset Object](#) | [Supports Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Microsoft OLE DB Provider for Microsoft Jet

The OLE DB Provider for Microsoft Jet allows ADO to access Microsoft Jet databases.

Connection String Parameters

To connect to this [provider](#), set the *Provider* argument of the [ConnectionString](#) property to:

```
Microsoft.Jet.OLEDB.4.0
```

Reading the [Provider](#) property will return this string as well.

Typical Connection String

A typical connection string for this provider is:

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=databaseName;User ID=u
```

The string consists of these keywords:

Keyword	Description
Provider	Specifies the OLE DB Provider for Microsoft Jet.
Data Source	Specifies the database path and file name (for example, c:\Northwind.mdb).
User ID	Specifies the user name. If this keyword is not specified, the string, "admin", is used by default.
Password	Specifies the user password. If this keyword is not specified, the empty string (""), is used by default.

Provider-Specific Connection Parameters

The OLE DB Provider for Microsoft Jet supports several provider-specific dynamic properties in addition to those defined by ADO. As with all other **Connection** parameters, they can be set via the **Connection** object's **Properties** collection or as part of the connection string.

The following table lists these properties with the corresponding OLE DB property name in parentheses.

Parameter	
Jet OLEDB:Compact Reclaimed Space Amount (DBPROP_JETOLEDB_COMPACTFREESPACE)	Indicates an es space, in bytes compacting the only valid afte been establishe
Jet OLEDB:Connection Control (DBPROP_JETOLEDB_CONNECTIONCONTROL)	Indicates whet the database.
Jet OLEDB:Create System Database (DBPROP_JETOLEDB_CREATESYSTEMDATABASE)	Indicates whet should be creat data source.
Jet OLEDB:Database Locking Mode (DBPROP_JETOLEDB_DATABASELOCKMODE)	Indicates the lc database. The 1 database deter the database is
Jet OLEDB:Database Password (DBPROP_JETOLEDB_DATABASEPASSWORD)	Indicates the d
Jet OLEDB:Don't Copy Locale on Compact (DBPROP_JETOLEDB_COMPACT_DONTCOPYLOCALE)	Indicates whet information wl
Jet OLEDB:Encrypt Database (DBPROP_JETOLEDB_ENCRYPTDATABASE)	Indicates whet should be enchr not set, the cor encrypted if th also encrypted
Jet OLEDB:Engine Type (DBPROP_JETOLEDB_ENGINE)	Indicates the st access the curr

Jet OLEDB:Exclusive Async Delay
(DBPROP_JETOLEDB_EXCLUSIVEASYNCDELAY)

Indicates the number of milliseconds, to wait before the asynchronous database is opened.

This property is only applicable to **OLEDB:Flus** and is set to 0.

Jet OLEDB:Flush Transaction Timeout
(DBPROP_JETOLEDB_FLUSHTRANSACTIONTIMEOUT)

Indicates the amount of time to wait before data stored in memory is flushed to disk. This setting applies to **Jet OLEDB** and **Jet OLEDB Async Delay**.

Jet OLEDB:Global Bulk Transactions
(DBPROP_JETOLEDB_GLOBALBULKNOTTRANSACTIONS)

Indicates whether global bulk transactions are transacted.

Jet OLEDB:Global Partial Bulk Ops
(DBPROP_JETOLEDB_GLOBALBULKPARTIAL)

Indicates the percentage of global bulk operations that are partial.

Jet OLEDB:Implicit Commit Sync
(DBPROP_JETOLEDB_IMPLICITCOMMITSYNC)

Indicates whether implicit commits are synchronous.

Jet OLEDB:Lock Delay
(DBPROP_JETOLEDB_LOCKDELAY)

Indicates the number of milliseconds to wait before attempting to acquire a lock after a previous attempt.

Jet OLEDB:Lock Retry
(DBPROP_JETOLEDB_LOCKRETRY)

Indicates how many times to attempt to access a locked resource.

Jet OLEDB:Max Buffer Size
(DBPROP_JETOLEDB_MAXBUFFERSIZE)

Indicates the maximum amount of memory, in kilobytes, that can be used before it starts flushing data to disk.

Jet OLEDB:Max Locks Per File
(DBPROP_JETOLEDB_MAXLOCKSPERFILE)

Indicates the maximum number of locks that Jet can place on a file. The default value is 9500.

Jet OLEDB:New Database Password
(DBPROP_JETOLEDB_NEWDATABASEPASSWORD)

Indicates the password for the new database. The default value is " " in **Jet OLEDB**.

Jet OLEDB:ODBC Command Time Out
(DBPROP_JETOLEDB_ODBCCOMMANDTIMEOUT)

Indicates the number of seconds before a remote server will timeout.

Jet OLEDB:Page Locks to Table Lock
(DBPROP_JETOLEDB_PAGELOCKSTOTABLELOCK)

Indicates how many pages are locked within a table. If the number of attempts to promote a page lock exceeds this value, the lock is never promoted to a table lock.

Jet OLEDB:Page Timeout
(DBPROP_JETOLEDB_PAGETIMEOUT)

Indicates the number of seconds the engine will wait before flushing the cache if it is out of sync.

Jet OLEDB:Recycle Long-Valued Pages
(DBPROP_JETOLEDB_RECYCLELONGVALUEPAGES)

Indicates whether the engine should try to reclaim long-valued pages when they are freed.

Jet OLEDB:Registry Path
(DBPROP_JETOLEDB_REGPATH)

Indicates the registry path that contains the values for the Jet engine.

Jet OLEDB:Reset ISAM Stats
(DBPROP_JETOLEDB_RESETISAMSTATS)

Indicates whether the engine should reset its ISAM statistics after returning a DBSCHEMA_CHANGED error.

Jet OLEDB:Shared Async Delay
(DBPROP_JETOLEDB_SHAREDASYNCDELAY)

Indicates the number of milliseconds the engine should wait for an asynchronous operation to complete before the database is opened.

Jet OLEDB:System Database
(DBPROP_JETOLEDB_SYSDBPATH)

Indicates the path to the system database (in a workgroup environment, this is the path to the system database).

Jet OLEDB:Transaction Commit Mode
(DBPROP_JETOLEDB_TXNCOMMITMODE)

Indicates whether the engine should commit a transaction synchronously or asynchronously.

Jet OLEDB>User Commit Sync
(DBPROP_JETOLEDB_USERCOMMITSYNC)

Indicates whether the engine should commit transactions asynchronously or synchronously.

Provider-Specific Recordset and Command Properties

The Jet provider also supports several provider-specific **Recordset** and **Command** properties. These properties are accessed and set through the **Properties** collection of the **Recordset** or **Command** object. The table lists the ADO property name and its corresponding OLE DB property name in parentheses.

Property Name	Description
Jet OLEDB:Bulk Transactions (DBPROP_JETOLEDB_BULKNOTTRANSACTIONS)	Indicates whether bulk operations are transactional. Large bulk operations may fail when transactions are used to resource delays.
Jet OLEDB:Enable Fat Cursors (DBPROP_JETOLEDB_ENABLEFATCURSOR)	Indicates whether multiple cursors should be opened when possible. A recordset from a remote data source.
Jet OLEDB:Fat Cursor Cache Size (DBPROP_JETOLEDB_FATCURSORMAXROWS)	Indicates the number of rows to cache using remote data storage caching. The value is 100 unless Jet OLEDB Fat Cursor

Jet OLEDB:Inconsistent
(DBPROP_JETOLEDB_INCONSISTENT)

Jet OLEDB:Locking Granularity
(DBPROP_JETOLEDB_LOCKGRANULARITY)

Jet OLEDB:ODBC Pass-Through Statement
(DBPROP_JETOLEDB_ODBCPASSTHROUGH)

Jet OLEDB:Partial Bulk Ops
(DBPROP_JETOLEDB_BULKPARTIAL)

Jet OLEDB:Pass Through Query Bulk-Op
(DBPROP_JETOLEDB_PASSTHROUGHBULKOP)

Jet OLEDB:Pass Through Query Connect String
(DBPROP_JETOLEDB_ODBCPASSTHROUGHCONNECTSTRING)

True.
Indicates whether results are inconsistent updates.
Indicates whether is opened row-level locking.
Indicates should pass SQL text through **Comma** object to back end unaltered.
Indicates behavior of SQL DM operation.
Indicates whether that do not return a **Records** passed up to the data source.
Indicates connect used to connect to a remote store. This is ignored by **Jet OLEDB**

Jet OLEDB:Stored Query
(DBPROP_JETOLEDB_STOREDQUERY)

Jet OLEDB:Validate Rules On Set
(DBPROP_JETOLEDB_VALIDATEONSET)

**Pass-Th
Stateme
True.**

Indicates
whether
comman
should b
interpret
stored q
instead c
SQL cor

Indicates
whether
validatic
are evalu
when co
data is s
when the
changes
committ
database

By default, the OLE DB Provider for Microsoft Jet opens Microsoft Jet databases in read/write mode. To open a database in read-only mode, set the [Mode](#) property on the ADO **Connection** object to **adModeRead**.

Command Object Usage

Command text in the [Command](#) object uses the Microsoft Jet SQL dialect. You can specify row-returning queries, action queries, and table names in the command text; however, stored procedures are not supported and should not be specified.

Recordset Behavior

The Microsoft Jet database engine does not support dynamic [cursors](#). Therefore, the OLE DB Provider for Microsoft Jet does not support the **adLockDynamic** cursor type. When a dynamic cursor is requested, the provider will return a keyset cursor and reset the [CursorType](#) property to indicate the type of [Recordset](#) returned. Further, if an updatable **Recordset** is requested (**LockType** is **adLockOptimistic**, **adLockBatchOptimistic**, or **adLockPessimistic**) the provider will also return a keyset cursor and reset the **CursorType** property.

Dynamic Properties

The OLE DB Provider for Microsoft Jet inserts several dynamic properties into the **Properties** collection of the unopened [Connection](#), [Recordset](#), and [Command](#) objects.

The tables below are a cross-index of the ADO and OLE DB names for each dynamic property. The OLE DB Programmer's Reference refers to an ADO property name by the term, "Description." You can find more information about these properties in the OLE DB Programmer's Reference. Search for the OLE DB property name in the Index or see Appendix C: OLE DB Properties.

Connection Dynamic Properties

The following properties are added to the **Connection** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Active Sessions	DBPROP_ACTIVESESSIONS
Asynchable Abort	DBPROP_ASYNCTXNABORT
Asynchable Commit	DBPROP_ASYNCTNXCOMMIT
Autocommit Isolation Levels	DBPROP_SESS_AUTOCOMMITISOLEVELS
Catalog Location	DBPROP_CATALOGLOCATION
Catalog Term	DBPROP_CATALOGTERM
Column Definition	DBPROP_COLUMNDEFINITION
Current Catalog	DBPROP_CURRENTCATALOG
Data Source	DBPROP_INIT_DATASOURCE
Data Source Name	DBPROP_DATASOURCENAME
Data Source Object Threading Model	DBPROP_DSOTHREADMODEL
DBMS Name	DBPROP_DBMSNAME
DBMS Version	DBPROP_DBMSVER
GROUP BY Support	DBPROP_GROUPBY
Heterogeneous Table Support	DBPROP_HETEROGENEOUSTABLES
Identifier Case Sensitivity	DBPROP_IDENTIFIER_CASE
Isolation Levels	DBPROP_SUPPORTEDTXNISOLEVELS
Isolation Retention	DBPROP_SUPPORTEDTXNISORETAIN
Locale Identifier	DBPROP_INIT_LCID
Maximum Index Size	DBPROP_MAXINDEXSIZE
Maximum Row Size	DBPROP_MAXROWSIZE
Maximum Row Size Includes BLOB	DBPROP_MAXROWSIZEINCLUDESBLOB

Maximum Tables in SELECT	DBPROP_MAXTABLESINSELECT
Mode	DBPROP_INIT_MODE
Multiple Parameter Sets	DBPROP_MULTIPLEPARAMSETS
Multiple Results	DBPROP_MULTIPLERESULTS
Multiple Storage Objects	DBPROP_MULTIPLESTORAGEOBJECTS
Multi-Table Update	DBPROP_MULTITABLEUPDATE
NULL Collation Order	DBPROP_NULLCOLLATION
NULL Concatenation Behavior	DBPROP_CONCATNULLBEHAVIOR
OLE DB Version	DBPROP_PROVIDEROLEDBVER
OLE Object Support	DBPROP_OLEOBJECTS
Open Rowset Support	DBPROP_OPENROWSETSUPPORT
ORDER BY Columns in Select List	DBPROP_ORDERBYCOLUMNSINSELECT
Output Parameter Availability	DBPROP_OUTPUTPARAMETERAVAILABILITY
Pass By Ref Accessors	DBPROP_BYREFACCESSORS
Password	DBPROP_AUTH_PASSWORD
Persistent ID Type	DBPROP_PERSISTENTIDTYPE
Prepare Abort Behavior	DBPROP_PREPAREABORTBEHAVIOR
Prepare Commit Behavior	DBPROP_PREPARECOMMITBEHAVIOR
Procedure Term	DBPROP_PROCEDURETERM
Prompt	DBPROP_INIT_PROMPT
Provider Friendly Name	DBPROP_PROVIDERFRIENDLYNAME
Provider Name	DBPROP_PROVIDERFILENAME
Provider Version	DBPROP_PROVIDERVER
Read-Only Data Source	DBPROP_DATASOURCEADONLY
Rowset Conversions on Command	DBPROP_ROWSETCONVERSIONSONCOMMAND

Schema Term	DBPROP_SCHEMATERM
Schema Usage	DBPROP_SCHEMAUSAGE
SQL Support	DBPROP_SQLSUPPORT
Structured Storage	DBPROP_STRUCTUREDSTORAGE
Subquery Support	DBPROP_SUBQUERIES
Table Term	DBPROP_TABLETERM
Transaction DDL	DBPROP_SUPPORTEDTXNDDL
User ID	DBPROP_AUTH_USERID
User Name	DBPROP_USERNAME
Window Handle	DBPROP_INIT_HWND

Recordset Dynamic Properties

The following properties are added to the **Recordset** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Access Order	DBPROP_ACCESSORDER
Append-Only Rowset	DBPROP_APPENDONLY
Blocking Storage Objects	DBPROP_BLOCKINGSTORAGEOBJECTS
Bookmark Type	DBPROP_BOOKMARKTYPE
Bookmarkable	DBPROP_IROWSETLOCATE
Bookmarks Ordered	DBPROP_ORDEREDBOOKMARKS
Cache Deferred Columns	DBPROP_CACHEDEFERRED
Change Inserted Rows	DBPROP_CHANGEINSERTEDROWS
Column Privileges	DBPROP_COLUMNRESTRICT
Column Set Notification	DBPROP_NOTIFYCOLUMNSET
Column Writable	DBPROP_MAYWRITECOLUMN
Defer Column	DBPROP_DEFERRED
Delay Storage Object Updates	DBPROP_DELAYSTORAGEOBJECTS
Fetch Backwards	DBPROP_CANFETCHBACKWARDS
Hold Rows	DBPROP_CANHOLDROWS
IAccessor	DBPROP_IAccessor
IColumnsInfo	DBPROP_IColumnsInfo
IColumnsRowset	DBPROP_IColumnsRowset
IConnectionPointContainer	DBPROP_IConnectionPointContainer
IConvertType	DBPROP_IConvertType
ILockBytes	DBPROP_ILockBytes
Immobile Rows	DBPROP_IMMOBILEROWS
IRowset	DBPROP_IRowset
IRowsetChange	DBPROP_IRowsetChange
IRowsetIdentity	DBPROP_IRowsetIdentity
IRowsetIndex	DBPROP_IRowsetIndex

IRowsetInfo	DBPROP_IRowsetInfo
IRowsetLocate	DBPROP_IRowsetLocate
IRowsetResynch	
IRowsetScroll	DBPROP_IRowsetScroll
IRowsetUpdate	DBPROP_IRowsetUpdate
ISequentialStream	DBPROP_ISequentialStream
IStorage	DBPROP_IStorage
IStream	DBPROP_IStream
ISupportErrorInfo	DBPROP_ISupportErrorInfo
Literal Bookmarks	DBPROP_LITERALBOOKMARKS
Literal Row Identity	DBPROP_LITERALIDENTITY
Maximum Open Rows	DBPROP_MAXOPENROWS
Maximum Pending Rows	DBPROP_MAXPENDINGROWS
Maximum Rows	DBPROP_MAXROWS
Memory Usage	DBPROP_MEMORYUSAGE
Notification Granularity	DBPROP_NOTIFICATIONGRANULARITY
Notification Phases	DBPROP_NOTIFICATIONPHASES
Objects Transacted	DBPROP_TRANSACTEDOBJECT
Others' Changes Visible	DBPROP_OTHERUPDATEDELETE
Others' Inserts Visible	DBPROP_OTHERINSERT
Own Changes Visible	DBPROP_OWNUPDATEDELETE
Own Inserts Visible	DBPROP_OWNINSERT
Preserve on Abort	DBPROP_ABORTPRESERVE
Preserve on Commit	DBPROP_COMMITPRESERVE
Quick Restart	DBPROP_QUICKRESTART
Reentrant Events	DBPROP_REENTRANTEVENTS
Remove Deleted Rows	DBPROP_REMOVEDELETED
Report Multiple Changes	DBPROP_REPORTMULTIPLECHANGES
Return Pending Inserts	DBPROP_RETURNPENDINGINSERTS
Row Delete Notification	DBPROP_NOTIFYROWDELETE
Row First Change Notification	DBPROP_NOTIFYROWFIRSTCHANGE
Row Insert Notification	DBPROP_NOTIFYROWINSERT

Row Privileges	DBPROP_ROWRESTRICT
Row Resynchronization Notification	DBPROP_NOTIFYROWRESYNCH
Row Threading Model	DBPROP_ROWTHREADMODEL
Row Undo Change Notification	DBPROP_NOTIFYROWUNDOCHANGE
Row Undo Delete Notification	DBPROP_NOTIFYROWUNDODELETE
Row Undo Insert Notification	DBPROP_NOTIFYROWUNDOINSERT
Row Update Notification	DBPROP_NOTIFYROWUPDATE
Rowset Fetch Position Change Notification	DBPROP_NOTIFYROWSETFETCHPOSITIONCH
Rowset Release Notification	DBPROP_NOTIFYROWSETRELEASE
Scroll Backwards	DBPROP_CANSROLLBACKWARDS
Skip Deleted Bookmarks	DBPROP_BOOKMARKSKIPPED
Strong Row Identity	DBPROP_STRONGIDENTITY
Updatability	DBPROP_UPDATABILITY
Use Bookmarks	DBPROP_BOOKMARKS

Command Dynamic Properties

The following properties are added to the **Command** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Access Order	DBPROP_ACCESSORDER
Append-Only Rowset	DBPROP_APPENDONLY
Blocking Storage Objects	DBPROP_BLOCKINGSTORAGEOBJECTS
Bookmark Type	DBPROP_BOOKMARKTYPE
Bookmarkable	DBPROP_IROWSETLOCATE
Change Inserted Rows	DBPROP_CHANGEINSERTEDROWS
Column Privileges	DBPROP_COLUMNRESTRICT
Column Set Notification	DBPROP_NOTIFYCOLUMNSET
Defer Column	DBPROP_DEFERRED
Delay Storage Object Updates	DBPROP_DELAYSTORAGEOBJECTS
Fetch Backwards	DBPROP_CANFETCHBACKWARDS
Hold Rows	DBPROP_CANHOLDROWS
IAccessor	DBPROP_IAccessor
IColumnsInfo	DBPROP_IColumnsInfo
IColumnsRowset	DBPROP_IColumnsRowset
IConnectionPointContainer	DBPROP_IConnectionPointContainer
IConvertType	DBPROP_IConvertType
ILockBytes	DBPROP_ILockBytes
Immobile Rows	DBPROP_IMMOBILEROWS
IRowset	DBPROP_IRowset
IRowsetChange	DBPROP_IRowsetChange
IRowsetIdentity	DBPROP_IRowsetIdentity
IRowsetIndex	DBPROP_IRowsetIndex
IRowsetInfo	DBPROP_IRowsetInfo
IRowsetLocate	DBPROP_IRowsetLocate
IRowsetResynch	

IRowsetScroll	DBPROP_IRowsetScroll
IRowsetUpdate	DBPROP_IRowsetUpdate
ISequentialStream	DBPROP_ISequentialStream
IStorage	DBPROP_IStorage
IStream	DBPROP_IStream
ISupportErrorInfo	DBPROP_ISupportErrorInfo
Literal Bookmarks	DBPROP_LITERALBOOKMARKS
Literal Row Identity	DBPROP_LITERALIDENTITY
Lock Mode	DBPROP_LOCKMODE
Maximum Open Rows	DBPROP_MAXOPENROWS
Maximum Pending Rows	DBPROP_MAXPENDINGROWS
Maximum Rows	DBPROP_MAXROWS
Notification Granularity	DBPROP_NOTIFICATIONGRANULARITY
Notification Phases	DBPROP_NOTIFICATIONPHASES
Objects Transacted	DBPROP_TRANSACTEDOBJECT
Others' Changes Visible	DBPROP_OTHERUPDATEDELETE
Others' Inserts Visible	DBPROP_OTHERINSERT
Own Changes Visible	DBPROP_OWNUPDATEDELETE
Own Inserts Visible	DBPROP_OWNINSERT
Preserve on Abort	DBPROP_ABORTPRESERVE
Preserve on Commit	DBPROP_COMMITPRESERVE
Quick Restart	DBPROP_QUICKRESTART
Reentrant Events	DBPROP_REENTRANTEVENTS
Remove Deleted Rows	DBPROP_REMOVEDELETED
Report Multiple Changes	DBPROP_REPORTMULTIPLECHANGES
Return Pending Inserts	DBPROP_RETURNPENDINGINSERTS
Row Delete Notification	DBPROP_NOTIFYROWDELETE
Row First Change Notification	DBPROP_NOTIFYROWFIRSTCHANGE
Row Insert Notification	DBPROP_NOTIFYROWINSERT
Row Privileges	DBPROP_ROWRESTRICT
Row Resynchronization Notification	DBPROP_NOTIFYROWRESYNCH

Row Threading Model	DBPROP_ROWTHREADMODEL
Row Undo Change Notification	DBPROP_NOTIFYROWUNDOCHANGE
Row Undo Delete Notification	DBPROP_NOTIFYROWUNDODELETE
Row Undo Insert Notification	DBPROP_NOTIFYROWUNDOINSERT
Row Update Notification	DBPROP_NOTIFYROWUPDATE
Rowset Fetch Position Change Notification	DBPROP_NOTIFYROWSETFETCHPOSITIONCH
Rowset Release Notification	DBPROP_NOTIFYROWSETRELEASE
Scroll Backwards	DBPROP_CANSROLLBACKWARDS
Server Data on Insert	DBPROP_SERVERDATAONINSERT
Skip Deleted Bookmarks	DBPROP_BOOKMARKSKIP
Strong Row Identity	DBPROP_STRONGIDENTITY
Updatability	DBPROP_UPDATABILITY
Use Bookmarks	DBPROP_BOOKMARKS

See Also For specific implementation details and functional information about the OLE DB Provider for Microsoft Jet, consult the OLE DB Provider for Microsoft Jet documentation in the MDAC SDK.

See Also

[ConnectionString Property](#) | [Provider Property](#) | [Recordset Object](#)

OLE DB Provider for Microsoft Jet in the OLE DB documentation.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Microsoft OLE DB Provider for SQL Server

The Microsoft OLE DB Provider for SQL Server, SQLOLEDB, allows ADO to access Microsoft SQL Server.

Connection String Parameters

To connect to this [provider](#), set the *Provider* argument to the [ConnectionString](#) property to:

SQLOLEDB

This value can also be set or read using the [Provider](#) property.

Typical Connection String

A typical connection string for this provider is:

```
"Provider=SQLOLEDB;Data Source=serverName;"  
Initial Catalog=databaseName;  
User ID=userName;Password=userPassword;"
```

The string consists of these keywords:

Keyword	Description
Provider	Specifies the OLE DB Provider for SQL Server.
Data Source or Server	Specifies the name of a server.
Initial Catalog or Database	Specifies the name of a database on the server.
User ID or uid	Specifies the user name (for SQL Server Authentication).
Password or pwd	Specifies the user password (for SQL Server Authentication).

Provider-Specific Connection Parameters

The provider 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 via the [Properties](#) collection of a [Connection](#) or can be set as part of the **ConnectionString**.

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 , then SQLOLEDB uses Microsoft Windows NT Authentication Mode to authorize user access to the SQL Server database specified by the Location and Datasource property values. If this property is set to No , then 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 language used for system message selection and formatting. The language must be installed on the SQL Server, otherwise opening the connection will fail.
Network Address	Indicates the network address of the 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 Prepare	Determines whether SQL Server creates temporary stored procedures when Commands are prepared (by the Prepared property).
	Indicates whether OEM/ANSI characters are converted. This property can be set to True or

Auto Translate	<p>False. The default value is True. If this property is set to True, then SQLOLEDB performs OEM/ANSI character conversion when multi-byte character strings are retrieved from, or sent to, the SQL Server. If this property is set to False, then SQLOLEDB does not perform OEM/ANSI character conversion on multi-byte character string data.</p>
Packet Size	<p>Indicates a network packet size in bytes. The packet size property value must be between 512 and 32767. The default SQLOLEDB network packet size is 4096.</p>
Application Name	<p>Indicates the client application name.</p>
Workstation ID	<p>A string identifying the workstation.</p>

Command Object Usage

SQLOLEDB accepts an amalgam of ODBC, ANSI, and SQL Server-specific Transact-SQL as valid syntax. For example, the following SQL statement uses an [ODBC](#) SQL escape sequence to specify the LCASE string function:

```
SELECT customerid={fn LCASE(CustomerID)} FROM Customers
```

LCASE returns a character string, converting all uppercase characters to their lowercase equivalents. The ANSI SQL string function LOWER performs the same operation, so the following SQL statement is an ANSI equivalent to the ODBC statement presented above:

```
SELECT customerid=LOWER(CustomerID) FROM Customers
```

SQLOLEDB successfully processes either form of the statement when specified as text for a command.

Stored Procedures

When executing a SQL Server stored procedure using a SQLOLEDB command, use the ODBC procedure call escape sequence in the command text.

SQLOLEDB then uses the remote procedure call mechanism of SQL Server to optimize command processing. For example, the following ODBC SQL statement is the preferred command text over the Transact-SQL form:

ODBC SQL

```
{call SalesByCategory('Produce', '1995')}
```

Transact-SQL

```
EXECUTE SalesByCategory 'Produce', '1995'
```

Recordset Behavior

SQLOLEDB cannot use SQL Server [cursors](#) to support the multiple-result generated by many commands. If a consumer requests a recordset requiring SQL Server cursor support, an error occurs if the command text used generates more than a single recordset as its result.

Scrollable SQLOLEDB recordsets are supported by SQL Server cursors. SQL Server imposes limitations on cursors that are sensitive to changes made by other users of the database. Specifically, the rows in some cursors cannot be ordered, and attempting to create a recordset using a command containing an SQL `ORDER BY` clause can fail.

Dynamic Properties

The Microsoft OLE DB Provider for SQL Server inserts several dynamic properties into the **Properties** collection of the unopened [Connection](#), [Recordset](#), and [Command](#) objects.

The following tables are a cross-index of the ADO and OLE DB names for each dynamic property. The OLE DB Programmer's Reference refers to an ADO property name by the term "Description." You can find more information about these properties in the OLE DB Programmer's Reference. Search for the OLE DB property name in the Index or see Appendix C: OLE DB Properties.

Connection Dynamic Properties

The following properties are added to the **Connection** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Active Sessions	DBPROP_ACTIVESESSIONS
Asynchable Abort	DBPROP_ASYNCTXNABORT
Asynchable Commit	DBPROP_ASYNCTNXCOMMIT
Autocommit Isolation Levels	DBPROP_SESS_AUTOCOMMITISOLEVELS
Catalog Location	DBPROP_CATALOGLOCATION
Catalog Term	DBPROP_CATALOGTERM
Column Definition	DBPROP_COLUMNDEFINITION
Connect Timeout	DBPROP_INIT_TIMEOUT
Current Catalog	DBPROP_CURRENTCATALOG
Data Source	DBPROP_INIT_DATASOURCE
Data Source Name	DBPROP_DATASOURCENAME
Data Source Object Threading Model	DBPROP_DSOTHREADMODEL
DBMS Name	DBPROP_DBMSNAME
DBMS Version	DBPROP_DBMSVER
Extended Properties	DBPROP_INIT_PROVIDERSTRING
GROUP BY Support	DBPROP_GROUPBY
Heterogeneous Table Support	DBPROP_HETEROGENEOUSTABLES
Identifier Case Sensitivity	DBPROP_IDENTIFIER_CASE
Initial Catalog	DBPROP_INIT_CATALOG
Isolation Levels	DBPROP_SUPPORTEDTXNISOLEVELS
Isolation Retention	DBPROP_SUPPORTEDTXNISORETAIN
Locale Identifier	DBPROP_INIT_LCID
Maximum Index Size	DBPROP_MAXINDEXSIZE

Maximum Row Size	DBPROP_MAXROWSIZE
Maximum Row Size Includes BLOB	DBPROP_MAXROWSIZEINCLUDESBLOB
Maximum Tables in SELECT	DBPROP_MAXTABLESINSELECT
Multiple Parameter Sets	DBPROP_MULTIPLEPARAMSETS
Multiple Results	DBPROP_MULTIPLERESULTS
Multiple Storage Objects	DBPROP_MULTIPLESTORAGEOBJECTS
Multi-Table Update	DBPROP_MULTITABLEUPDATE
NULL Collation Order	DBPROP_NULLCOLLATION
NULL Concatenation Behavior	DBPROP_CONCATNULLBEHAVIOR
OLE DB Version	DBPROP_PROVIDEROLEDBVER
OLE Object Support	DBPROP_OLEOBJECTS
Open Rowset Support	DBPROP_OPENROWSETSSUPPORT
ORDER BY Columns in Select List	DBPROP_ORDERBYCOLUMNSINSELECT
Output Parameter Availability	DBPROP_OUTPUTPARAMETERAVAILABILITY
Pass By Ref Accessors	DBPROP_BYREFACCESSORS
Password	DBPROP_AUTH_PASSWORD
Persist Security Info	DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO
Persistent ID Type	DBPROP_PERSISTENTIDTYPE
Prepare Abort Behavior	DBPROP_PREPAREABORTBEHAVIOR
Prepare Commit Behavior	DBPROP_PREPARECOMMITBEHAVIOR
Procedure Term	DBPROP_PROCEDURETERM
Prompt	DBPROP_INIT_PROMPT
Provider Friendly Name	DBPROP_PROVIDERFRIENDLYNAME
Provider Name	DBPROP_PROVIDERFILENAME

Provider Version	DBPROP_PROVIDERSERVER
Read-Only Data Source	DBPROP_DATASOURCEREADONLY
Rowset Conversions on Command	DBPROP_ROWSETCONVERSIONSONCOMMAND
Schema Term	DBPROP_SCHEMATERM
Schema Usage	DBPROP_SCHEMAUSAGE
SQL Support	DBPROP_SQLSUPPORT
Structured Storage	DBPROP_STRUCTUREDSTORAGE
Subquery Support	DBPROP_SUBQUERIES
Table Term	DBPROP_TABLETERM
Transaction DDL	DBPROP_SUPPORTEDTXNDDL
User ID	DBPROP_AUTH_USERID
User Name	DBPROP_USERNAME
Window Handle	DBPROP_INIT_HWND

Recordset Dynamic Properties

The following properties are added to the **Recordset** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Access Order	DBPROP_ACCESSORDER
Blocking Storage Objects	DBPROP_BLOCKINGSTORAGEOBJECTS
Bookmark Type	DBPROP_BOOKMARKTYPE
Bookmarkable	DBPROP_IROWSETLOCATE
Change Inserted Rows	DBPROP_CHANGEINSERTEDROWS
Column Privileges	DBPROP_COLUMNRESTRICT
Column Set Notification	DBPROP_NOTIFYCOLUMNSET
Command Time Out	DBPROP_COMMANDTIMEOUT
Defer Column	DBPROP_DEFERRED
Delay Storage Object Updates	DBPROP_DELAYSTORAGEOBJECTS
Fetch Backwards	DBPROP_CANFETCHBACKWARDS
Hold Rows	DBPROP_CANHOLDROWS
IAccessor	DBPROP_IAccessor
IColumnsInfo	DBPROP_IColumnsInfo
IColumnsRowset	DBPROP_IColumnsRowset
IConnectionPointContainer	DBPROP_IConnectionPointContainer
IConvertType	DBPROP_IConvertType
Immobile Rows	DBPROP_IMMOBILEROWS
IRowset	DBPROP_IRowset
IRowsetChange	DBPROP_IRowsetChange
IRowsetIdentity	DBPROP_IRowsetIdentity
IRowsetInfo	DBPROP_IRowsetInfo
IRowsetLocate	DBPROP_IRowsetLocate
IRowsetResynch	
IRowsetScroll	DBPROP_IRowsetScroll
IRowsetUpdate	DBPROP_IRowsetUpdate

ISequentialStream	DBPROP_ISequentialStream
ISupportErrorInfo	DBPROP_ISupportErrorInfo
Literal Bookmarks	DBPROP_LITERALBOOKMARKS
Literal Row Identity	DBPROP_LITERALIDENTITY
Maximum Open Rows	DBPROP_MAXOPENROWS
Maximum Pending Rows	DBPROP_MAXPENDINGROWS
Maximum Rows	DBPROP_MAXROWS
Notification Granularity	DBPROP_NOTIFICATIONGRANULARITY
Notification Phases	DBPROP_NOTIFICATIONPHASES
Objects Transacted	DBPROP_TRANSACTEDOBJECT
Others' Changes Visible	DBPROP_OTHERUPDATEDELETE
Others' Inserts Visible	DBPROP_OTHERINSERT
Own Changes Visible	DBPROP_OWNUPDATEDELETE
Own Inserts Visible	DBPROP_OWNINSERT
Preserve on Abort	DBPROP_ABORTPRESERVE
Preserve on Commit	DBPROP_COMMITPRESERVE
Quick Restart	DBPROP_QUICKRESTART
Reentrant Events	DBPROP_REENTRANTEVENTS
Remove Deleted Rows	DBPROP_REMOVEDELETED
Report Multiple Changes	DBPROP_REPORTMULTIPLECHANGES
Return Pending Inserts	DBPROP_RETURNPENDINGINSERTS
Row Delete Notification	DBPROP_NOTIFYROWDELETE
Row First Change Notification	DBPROP_NOTIFYROWFIRSTCHANGE
Row Insert Notification	DBPROP_NOTIFYROWINSERT
Row Privileges	DBPROP_ROWRESTRICT
Row Resynchronization Notification	DBPROP_NOTIFYROWRESYNCH
Row Threading Model	DBPROP_ROWTHREADMODEL
Row Undo Change Notification	DBPROP_NOTIFYROWUNDOCHANGE
Row Undo Delete Notification	DBPROP_NOTIFYROWUNDODELETE
Row Undo Insert	

Notification	DBPROP_NOTIFYROWUNDOINSERT
Row Update Notification	DBPROP_NOTIFYROWUPDATE
Rowset Fetch Position Change Notification	DBPROP_NOTIFYROWSETFETCHPOSITIONCH
Rowset Release Notification	DBPROP_NOTIFYROWSETRELEASE
Scroll Backwards	DBPROP_CANSCROLLBACKWARDS
Server Cursor	DBPROP_SERVERCURSOR
Skip Deleted Bookmarks	DBPROP_BOOKMARKSKIPPED
Strong Row Identity	DBPROP_STRONGIDENTITY
Unique Rows	DBPROP_UNIQUEROWS
Updatability	DBPROP_UPDATABILITY
Use Bookmarks	DBPROP_BOOKMARKS

Command Dynamic Properties

The following properties are added to the **Command** object's **Properties** collection.

ADO Property Name	OLE DB Property Name
Access Order	DBPROP_ACCESSORDER
Base Path	SSPROP_STREAM_BASEPATH
Blocking Storage Objects	DBPROP_BLOCKINGSTORAGEOBJECTS
Bookmark Type	DBPROP_BOOKMARKTYPE
Bookmarkable	DBPROP_IROWSETLOCATE
Change Inserted Rows	DBPROP_CHANGEINSERTEDROWS
Column Privileges	DBPROP_COLUMNRESTRICT
Column Set Notification	DBPROP_NOTIFYCOLUMNSET
Content Type	SSPROP_STREAM_CONTENTTYPE
Cursor Auto Fetch	SSPROP_CURSORAUTOFETCH
Defer Column	DBPROP_DEFERRED
Defer Prepare	SSPROP_DEFERPREPARE
Delay Storage Object Updates	DBPROP_DELAYSTORAGEOBJECTS
Fetch Backwards	DBPROP_CANFETCHBACKWARDS
Hold Rows	DBPROP_CANHOLDROWS
IAccessor	DBPROP_IAccessor
IColumnsInfo	DBPROP_IColumnsInfo
IColumnsRowset	DBPROP_IColumnsRowset
IConnectionPointContainer	DBPROP_IConnectionPointContainer
IConvertType	DBPROP_IConvertType
Immobile Rows	DBPROP_IMMOBILEROWS
IRowset	DBPROP_IRowset
IRowsetChange	DBPROP_IRowsetChange
IRowsetIdentity	DBPROP_IRowsetIdentity
IRowsetInfo	DBPROP_IRowsetInfo
IRowsetLocate	DBPROP_IRowsetLocate

IRowsetResynch	DBPROP_IRowsetResynch
IRowsetScroll	DBPROP_IRowsetScroll
IRowsetUpdate	DBPROP_IRowsetUpdate
ISequentialStream	DBPROP_ISequentialStream
ISupportErrorInfo	DBPROP_ISupportErrorInfo
Literal Bookmarks	DBPROP_LITERALBOOKMARKS
Literal Row Identity	DBPROP_LITERALIDENTITY
Lock Mode	DBPROP_LOCKMODE
Maximum Open Rows	DBPROP_MAXOPENROWS
Maximum Pending Rows	DBPROP_MAXPENDINGROWS
Maximum Rows	DBPROP_MAXROWS
Notification Granularity	DBPROP_NOTIFICATIONGRANULARITY
Notification Phases	DBPROP_NOTIFICATIONPHASES
Objects Transacted	DBPROP_TRANSACTEDOBJECT
Others' Changes Visible	DBPROP_OTHERUPDATEDELETE
Others' Inserts Visible	DBPROP_OTHERINSERT
Output Encoding Property	DBPROP_OUTPUTENCODING
Output Stream Property	DBPROP_OUTPUTSTREAM
Own Changes Visible	DBPROP_OWNUPLICATEDELETE
Own Inserts Visible	DBPROP_OWNINGINSERT
Preserve on Abort	DBPROP_ABORTPRESERVE
Preserve on Commit	DBPROP_COMMITPRESERVE
Quick Restart	DBPROP_QUICKRESTART
Reentrant Events	DBPROP_REENTRANTEVENTS
Remove Deleted Rows	DBPROP_REMOVEDELETED
Report Multiple Changes	DBPROP_REPORTMULTIPLECHANGES
Return Pending Inserts	DBPROP_RETURNPENDINGINSERTS
Row Delete Notification	DBPROP_NOTIFYROWDELETE
Row First Change Notification	DBPROP_NOTIFYROWFIRSTCHANGE
Row Insert Notification	DBPROP_NOTIFYROWINSERT
Row Privileges	DBPROP_ROWRESTRICT
Row Resynchronization	DBPROP_NOTIFYROWRESYNCH

Notification	
Row Threading Model	DBPROP_ROWTHREADMODEL
Row Undo Change Notification	DBPROP_NOTIFYROWUNDOCHANGE
Row Undo Delete Notification	DBPROP_NOTIFYROWUNDODELETE
Row Undo Insert Notification	DBPROP_NOTIFYROWUNDOINSERT
Row Update Notification	DBPROP_NOTIFYROWUPDATE
Rowset Fetch Position Change Notification	DBPROP_NOTIFYROWSETFETCHPOSITIONCH
Rowset Release Notification	DBPROP_NOTIFYROWSETRELEASE
Scroll Backwards	DBPROP_CANSROLLBACKWARDS
Server Cursor	DBPROP_SERVERCURSOR
Server Data on Insert	DBPROP_SERVERDATAONINSERT
Skip Deleted Bookmarks	DBPROP_BOOKMARKSKIP
Strong Row Identity	DBPROP_STRONGIDENTITY
Updatability	DBPROP_UPDATABILITY
Use Bookmarks	DBPROP_BOOKMARKS
XML Root	SSPROP_STREAM_XMLROOT
XSL	SSPROP_STREAM_XSL

For specific implementation details and functional information about the Microsoft SQL Server OLE DB Provider, consult the OLE DB Provider for SQL Server documentation in the OLE DB section of the MDAC SDK.

See Also

[ConnectionString Property](#) | [Provider Property](#) | [Recordset Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes 

Microsoft OLE DB Provider for Oracle

The Microsoft OLE DB Provider for Oracle allows ADO to access Oracle databases.

Connection String Parameters

To connect to this provider, set the *Provider* argument of the [ConnectionString](#) property to:

MSDAORA

Reading the [Provider](#) property will return this string as well.

If a join query with a keyset or dynamic [cursor](#) is executed in an Oracle database, an error occurs. Oracle only supports a static read-only cursor.

Typical Connection String

A typical connection string for this provider is:

```
"Provider=MSDAORA;Data Source=serverName;User ID=userName; Password=
```

The string consists of these keywords:

Keyword	Description
Provider	Specifies the OLE DB Provider for Oracle.
Data Source	Specifies the name of a server.
User ID	Specifies the user name.
Password	Specifies the user password.

Provider-Specific Connection Parameters

The provider 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 via the [Properties](#) collection of a [Connection](#) or as part of the **ConnectionString**.

These parameters are fully described in the OLE DB Programmer's Reference. (The [ADO Dynamic Property Index](#) provides a cross-reference between these parameter names and the corresponding OLE DB properties.)

Parameter	Description
Window Handle	Indicates the window handle to use to prompt for additional information.
Locale Identifier	Indicates a unique 32-bit number (for example, 1033) that specifies preferences related to the user's language. These preferences indicate how dates and times are formatted, items are sorted alphabetically, strings are compared, and so on.
OLE DB Services	Indicates a bitmask that specifies OLE DB services to enable or disable.
Prompt	Indicates whether to prompt the user while a connection is being established.
Extended Properties	A string containing provider-specific, extended connection information. Use this property only for provider-specific connection information that cannot be described through the property mechanism.

See Also

[ConnectionString Property](#) | [Provider Property](#) | [Recordset Object](#)

OLE DB Provider for Oracle in the OLE DB section of the MDAC SDK documentation

ADO 2.5 Appendixes

Microsoft OLE DB Provider for Internet Publishing

The Microsoft OLE DB Provider for Internet Publishing allows ADO to access resources served by Microsoft FrontPage or Microsoft Internet Information Server. Resources include web source files such as HTML files, or Windows 2000 web folders.

Connection String Parameters

To connect to this provider, set the *Provider* argument of the [ConnectionString](#) property to:

MSDAIPP.DSO

This value can also be set or read using the [Provider](#) property.

Typical Connection String

A typical connection string for this provider is:

```
"Provider=MSDAIPP.DSO;Data Source=ResourceURL;User ID=userName;Passw
```

-or-

```
"URL=ResourceURL;User ID=userName;Password=userPassword;"
```

The string consists of these keywords:

Keyword	Description
Provider	Specifies the OLE DB Provider for Internet Publishing.
Data Source -or- URL	Specifies the URL of a file or directory published in a Web Folder.
User ID	Specifies the user name.
Password	Specifies the user password.

If you set the *ResourceURL* value from the "URL=" in the connection string to an invalid value, by default the Internet Publishing Provider raises a dialog box to prompt for a valid value. This is undesirable behavior for a component in the middle tier of an application, because it suspends program execution until the dialog box is cleared and the client appears to freeze because it has not received a response from the component.

Note If MSDAIPP.DSO is explicitly specified as the value of the provider, either with the *Provider* connection string keyword or the **Provider** property, you cannot use "URL=" in the connection string. If you do, an error will occur. Instead, simply specify the URL as shown in the topic [Using ADO with the OLE DB Provider for Internet Publishing](#).

See Also

[Internet Publishing Scenario](#) | [Using ADO with the OLE DB Provider for Internet Publishing](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Appendixes

Microsoft Data Shaping Service for OLE DB

The Microsoft Data Shaping Service for OLE DB service [provider](#) supports the construction of hierarchical (shaped) [Recordset](#) objects from a data provider.

Provider Keyword

To invoke the Data Shaping Service for OLE DB, specify the following keyword and value in the connection string.

```
"Provider=MSDataShape"
```

Dynamic Properties

When this service provider is invoked, the following [dynamic properties](#) are added to the [Connection](#) object's [Properties](#) collection.

Dynamic Property Name	Description
Unique Reshape Names	Indicates whether Recordset objects with duplicate values for their Reshape Name properties are allowed. If this dynamic property is True and a new Recordset is created with the same user-specified reshape name as an existing Recordset , then the new Recordset object's reshape name is modified to make it unique. If this property is False and a new Recordset is created with the same user-specified reshape name as the existing Recordset , both Recordset objects will have the same reshape name. Therefore, neither Recordset can be reshaped as long as both recordsets exist. The default value of the property is False .
Data Provider	Indicates the name of the provider that will supply rows to be shaped . This value may be NONE if a provider will not be used to supply rows.

You may also set writable dynamic properties by specifying their names as keywords in the connection string. For example, in Microsoft Visual Basic, set the **Data Provider** dynamic property to "MSDASQL" by specifying:

```
Dim cn as New ADODB.Connection  
cn.Open "Provider=MSDataShape;Data Provider=MSDASQL"
```

You may also set or retrieve a dynamic property by specifying its name as the index to the [Properties](#) property. For example, get and print the current value of the **Data Provider** dynamic property, then set a new value, like this:

```
Debug.Print cn.Properties("Data Provider")
cn.Properties("Data Provider") = "MSDASQL"
```

For more information about data shaping, see [Data Shaping](#).

See Also

Data Shaping Service for OLE DB (in the OLE DB section of the MDAC SDK documentation)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Microsoft OLE DB Persistence Provider

The Microsoft OLE DB Persistence Provider enables you to save a [Recordset](#) object into a file, and later restore that **Recordset** object from the file. Schema information, data, and pending changes are preserved.

You can save the **Recordset** in either the proprietary Advanced Data Table Gram (ADTG) format, or the open Extensible Markup Language (XML) format.

Provider Keyword

To invoke this provider, specify the following keyword and value in the connection string.

`"Provider=MSPersist"`

Errors

The following errors issued by this provider can be detected in your application.

Constant	Description
E_BADSTREAM	The file opened does not have a valid format (that is, the format is not ADTG or XML).
E_CANTPERSISTROWSET	The Recordset object saved has characteristics that prevent it from being stored.

Remarks

The Microsoft OLE DB Persistence Provider exposes no [dynamic properties](#).

Currently, only parameterized hierarchical **Recordset** objects cannot be saved.

For more information about persistently storing **Recordset** objects, see [Recordset Persistence](#).

When a stream is used to open a **Recordset**, there should be no parameters specified other than the *Source* parameter of the **Open** method.

See Also

OLE DB Persistence Provider in the OLE DB section of the MDAC SDK documentation

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Microsoft OLE DB Remoting Provider

The Microsoft OLE DB Remoting Provider enables a local user on a [client](#) machine to invoke data [providers](#) on a remote machine. Specify the data provider parameters for the remote machine as you would if you were a local user on the remote machine. Then specify the parameters used by the Remoting Provider to access the remote machine. The resulting effect is that you will access the remote machine as if you were a local user.

Provider Keyword

To invoke the OLE DB Remoting Provider, specify the following keyword and value in the connection string. (Note the blank space in the provider name.)

```
"Provider=MS Remote"
```

Additional Keywords

When this service provider is invoked, the following additional keywords are relevant.

Keyword	Description
Data Source	Specifies the name of the remote data source. It is passed to the OLE DB Remoting Provider for processing. This keyword is equivalent to the RDS.DataControl object's Connect property.

Dynamic Properties

When this service provider is invoked, the following [dynamic properties](#) are added to the [Connection](#) object's [Properties](#) collection.

Dynamic Property Name	Description
DFMode	<p>Indicates the DataFactory Mode. A string that specifies the desired version of the DataFactory object on the server. Set this property before opening a connection to request a particular version of the DataFactory. If the requested version is not available, an attempt will be made to use the preceding version. If there is no preceding version, an error will occur. If DFMode is less than the available version, an error will occur. This property is read-only after a connection is made.</p> <p>Can be one of the following valid string values:</p> <ul style="list-style-type: none">• "25"—Version 2.5 (Default)• "21"—Version 2.1• "20"—Version 2.0• "15"—Version 1.5
Command Properties	<p>Indicates values that will be added to the string of command (rowset) properties sent to the server by the MS Remote provider. The default value for this string is vt_empty.</p> <p>Indicates the actual version number of the DataFactory on the server. Check this property to see if the version requested in the DFMode property was</p>

honored.

Can be one of the following valid Long integer values:

- 25—Version 2.5 (Default)
- 21—Version 2.1
- 20—Version 2.0
- 15—Version 1.5

Adding "DFMode=20;" to your connection string when using the **MSRemote** provider can improve your server's performance when updating data. With this setting, the **RDSServer.DataFactory** object on the server uses a less resource-intensive mode. However, the following features are not available in this configuration:

- Using parameterized queries.
- Getting parameter or column information before calling the **Execute** method.
- Setting **Transact Updates** to **True**.
- Getting row status.
- Calling the **Resync** method.
- Refreshing (explicitly or automatically) via the **Update Resync** property.
- Setting **Command** or **Recordset** properties.
- Using **adCmdTableDirect**.

Indicates the name of a server-side customization program (or handler) that extends the functionality of the [RDSServer.DataFactory](#), and any parameters used by the handler, all

Current DFMode

Handler

Internet Timeout

separated by commas (","). A **String** value.

Indicates the maximum number of milliseconds to wait for a request to travel to and from the server. (The default is 5 minutes.)

Remote Provider

Indicates the name of the [data provider](#) to be used on the remote server.

Remote Server

Indicates the server name and communication protocol to be used by this connection. This property is equivalent to the [RDS.DataControl](#) object [Server](#) property.

Transact Updates

When set to True, this value indicates that when [UpdateBatch](#) is performed on the server, it will be done inside a transaction. The default value for this Boolean dynamic property is False.

You may also set writable dynamic properties by specifying their names as keywords in the connection string. For example, set the **Internet Timeout** dynamic property to five seconds by specifying:

```
Dim cn as New ADODB.Connection
cn.Open "Provider=MS Remote;Internet Timeout=5000"
```

You may also set or retrieve a dynamic property by specifying its name as the index to the **Properties** property. For example, get and print the current value of the **Internet Timeout** dynamic property, and then set a new value, like this:

```
Debug.Print cn.Properties("Internet Timeout")
cn.Properties("Internet Timeout") = 5000
```

Remarks

In ADO 2.0, the OLE DB Remoting Provider could only be specified in the *ActiveConnection* parameter of the [Recordset](#) object **Open** method. Starting with ADO 2.1, the provider may also be specified in the *ConnectionString* parameter of the [Connection](#) object **Open** method.

The equivalent of the **RDS.DataControl** object [SQL](#) property is not available. The [Recordset](#) object **Open** method *Source* argument is used instead.

Specifying "...;Remote Provider=MS Remote;..." would create a four-tier scenario. Scenarios beyond three tiers have not been tested and should not be needed.

Example

This example performs a query on the **authors** table of the **pubs** database on a server named, *YourServer*. The names of the [remote data source](#) and remote server are provided in the [Connection](#) object [Open](#) method, and the SQL query is specified in the [Recordset](#) object [Open](#) method. A **Recordset** object is returned, edited, and used to update the data source.

```
Dim rs as New ADODB.Recordset
Dim cn as New ADODB.Connection
cn.Open "Provider=MS Remote;Data Source=pubs;" & _
       "Remote Server=http://YourServer"
rs.Open "SELECT * FROM authors", cn
...           'Edit the recordset
rs.UpdateBatch 'Equivalent of RDS SubmitChanges
...
```

See Also

OLE DB Remoting Provider in the OLE DB section of the MDAC SDK documentation

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Microsoft Cursor Service for OLE DB

The Microsoft Cursor Service for OLE DB supplements the [cursor](#) support functions of data providers. As a result, the user perceives relatively uniform functionality from all [data providers](#).

The Cursor Service makes [dynamic properties](#) available and enhances the behavior of certain methods. For example, the [Optimize](#) dynamic property enables the creation of temporary indexes to facilitate certain operations, such as the [Find](#) method.

The Cursor Service enables support for batch updating in all cases. It also simulates more capable cursor types, such as dynamic cursors, when a data provider can only supply less capable cursors, such as static cursors.

Keyword

To invoke this service component, set the [Recordset](#) or [Connection](#) object's [CursorLocation](#) property to **adUseClient**.

```
connection.CursorLocation=adUseClient  
recordset.CursorLocation=adUseClient
```

Dynamic Properties

When the Cursor Service for OLE DB is invoked, the following dynamic properties are added to the **Recordset** object's [Properties](#) collection. The full list of **Connection** and **Recordset** object dynamic properties is listed in the [ADO Dynamic Property Index](#). The associated OLE DB property names, where appropriate, are included in parenthesis after the ADO property name.

Changes to some dynamic properties are not visible to the underlying data source after the Cursor Service has been invoked. For example, setting the *Command Time out* property on a **Recordset** will not be visible to the underlying data provider.

```
...
Recordset1.CursorLocation = adUseClient      'invokes cursor service
Recordset1.Open "authors", _
    "Provider=SQLOLEDB;Data Source=DBServer;User Id=usr;" & _
    "Password=pwd;Initial Catalog=pubs;", , adCmdTable
Recordset1.Properties.Item("Command Time out") = 50
' 'Command Time out' property on DBServer is still default (30).
...
```

If your application requires the Cursor Service, but you need to set dynamic properties on the underlying provider, set the properties before invoking the Cursor Service. Command object property settings are always passed to the underlying data provider regardless of cursor location. Therefore, you can also use a command object to set the properties at any time.

Note The dynamic property DBPROP_SERVERDATAONINSERT is not supported by the cursor service, even if it is supported by the underlying data provider.

Property Name

Description

For recordsets created with the Data Shaping Service, this value indicates how often calculated and aggregate columns are calculated. The default

Auto Recalc
(DBPROP_ADC_AUTORECALC)

(value=1) is to recalculate whenever the Data Shaping Service determines that the values have changed. If the value is 0, the calculated or aggregate columns are only calculated when the hierarchy is initially built.

Batch Size
(DBPROP_ADC_BATCHSIZE)

Indicates the number of update statements that can be batched before being sent to the data store. The more statements in a batch, the fewer round trips to the data store.

Cache Child Rows
(DBPROP_ADC_CACHECHILDROWS)

For recordsets created with the Data Shaping Service, this value indicates whether child recordsets are stored in a cache for later use.

Cursor Engine Version
(DBPROP_ADC_CEVER)

Indicates the version of the Cursor Service being used.

Maintain Change Status
(DBPROP_ADC_MAINTAINCHANGESTATUS)

Indicates the text of the command used for resynchronizing a one or more rows in a multiple table join.

[Optimize](#)

Indicates whether an index should be created. When set to **True**, authorizes the temporary creation of indexes to improve the execution of certain operations.

[Reshape Name](#)

Indicates the name of the **Recordset**. May be referenced within the

[Resync Command](#)

[Unique Catalog](#)

[Unique Schema](#)

[Unique Table](#)

Update Criteria
(DBPROP_ADC_UPDATECRITERIA)

[Update Resync](#)
(DBPROP_ADC_UPDATERESYNC)

current, or subsequent, [data shaping](#) commands.

Indicates a custom command string that is used by the [Resync](#) method when the [Unique Table](#) property is in effect.

Indicates the name of the database containing the table referenced in the **Unique Table** property.

Indicates the name of the owner of the table referenced in the **Unique Table** property.

Indicates the name of the one table in a **Recordset** created from multiple tables that may be modified by insertions, updates, or deletions.

Indicates which fields in the **WHERE** clause are used to handle collisions occurring during an update.

Indicates whether the **Resync** method is implicitly invoked after the [UpdateBatch](#) method (and its behavior), when the **Unique Table** property is in effect.

You may also set or retrieve a [dynamic property](#) by specifying its name as the index to the **Properties** collection. For example, get and print the current value of the [Optimize](#) dynamic property, then set a new value, like this:

```
Debug.Print rs.Properties("Optimize")
```

```
rs.Properties("Optimize") = True
```

Built-in Property Behavior

The Cursor Service for OLE DB also affects the behavior of certain built-in properties.

Property Name	Description
CursorType	Supplements the types of cursors that are available for a Recordset .
LockType	Supplements the types of locks available for a Recordset . Enables batch updates.
Sort	Specifies one or more field names that the Recordset is sorted on, and whether each field is sorted in ascending or descending order.

Method Behavior

The Cursor Service for OLE DB enables or affects the behavior of the [Field](#) object's [Append](#) method; and the **Recordset** object's [Open](#), [Resync](#), [UpdateBatch](#), and [Save](#) methods.

See Also

Cursor Service for OLE DB in the OLE DB section of the MDAC SDK documentation

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Appendix B: ADO Errors

See the following topics for more information about particular error messages:

- [ADO Error Codes](#)
- [DataControl Error Codes](#)
- [Internet Explorer Error Codes](#)
- [Internet Information Services Error Codes](#)

See Also

[ADO API Reference](#) | [ADO Collections](#) | [ADO Dynamic Properties](#) | [ADO Enumerated Constants](#) | [ADO Events](#) | [ADO Methods](#) | [ADO Object Model](#) | [ADO Objects](#) | [ADO Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

ADO Error Codes

In addition to the [provider](#) errors returned in the [Error](#) objects of the [Errors](#) collection, ADO itself can return errors to the exception-handling mechanism of your run-time environment. Use the error trapping mechanism your programming language, such as the **On Error** statement in Microsoft Visual Basic or the **try-catch** block in Microsoft Visual C++ or Microsoft Visual J++ to capture ADO errors.

For the list of ADO error codes, see [ErrorValueEnum](#).

See Also

[Error Object](#) | [Errors Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

DataControl Error Codes

The following table lists the [RDS.DataControl](#) object error codes. The positive decimal translation of the low two bytes, the negative decimal translation of the full error code, and the hexadecimal values are shown.

RDS.DataControl error codes	Number	Description
IDS_AsyncPending	4107 -2146824175 0x800A1011	Operation cannot be performed while async operation is pending.
IDS_BadInlineTablegram	4105 -2146824183 0x800A1009	Bad inline tablegram.
IDS_CantConnect	4099 -2146824189 0x800A1003	Cannot connect to server.
IDS_CantCreateObject	4100 -2146824188 0x800A1004	Business object cannot be created.
IDS_CantFindDataspaces	4102 -2146824186 0x800A1006	Dataspaces property is not valid.
IDS_CantInvokeMethod	4101 -2146824187 0x800A1005	Method cannot be invoked on business object.
IDS_CrossDomainWarning	4112 -2146824170 0x800A1016	This page accesses data on another domain. Do you want to allow this? To avoid this message in Internet Explorer, you can add a secure Web site to your Trusted Sites zone on the Security tab of the Internet Options dialog box.
IDS_InvalidADCClientVersion	4106 -2146824176	Invalid RDS Client Version —

	0x800A1010	Client is newer than server.
IDS_INVALIDARG	5376 -2147019520 0x80071500	One or more arguments are invalid.
IDS_InvalidBindings	4097 -2146824191 0x800A1001	Error in bindings property.
IDS_InvalidParam	4110 -2146824172 0x800A1014	One or more arguments are invalid.
IDS_NOINTERFACE	5377 -2147019519 0x80071501	No such interface is supported.
IDS_NotReentrant	4111 -2146824171 0x800A1015	Request cannot be executed while the event handler is still processing.
IDS_ObjectNotSafe	4103 -2146824185 0x800A1007	Safety settings on this computer prohibit creation of business object.
IDS_RecordsetNotOpen	4109 -2146824173 0x800A1013	Recordset is not open.
IDS_ResetInvalidField	4108 -2146824174 0x800A1012	Column specified in SortColumn or FilterColumn does not exist.
IDS_RowsetNotUpdateable	4104 -2146824184 0x800A1008	Rowset not updateable.
IDS_UnexpectedError	4351 -2146823937 0x800A10FF	Unexpected error.
IDS_UpdatesFailed	4098 -2146824190 0x800A1002	Unable to update database.
	4119 -2146824169	DataControl URL property requires the system file

IDS_URLMONNotFound 0x800A1017 Urlmon.dll, which cannot be found.

See Also

[DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Internet Explorer Error Codes

The following table lists Microsoft Internet Explorer error codes related to Remote Data Service usage. The positive decimal translation of the low two bytes, the negative decimal translation of the full error code, and the hexadecimal values are shown.

Internet Explorer (Wininet) errors	Number	Description
IDS_WinInet_CantConnect	8195 -2146820093 0x800A2003	Internet Client Error: Cannot Connect to Server.
IDS_WinInet_ConnectionReset	12031 -2146816257 0x800A2EFF	Internet Client Error: Connection Reset.
IDS_WinInet_Error	8193 -2146820095 0x800A2001	Internet Client Error.
IDS_WinInet_InvalidServerResponse	8430 -2146819858 0x800A20EE	Internet Client Error: Invalid Server Response.
IDS_WinInet_SSLPostLimitation	8196 -2146820092 0x800A2004	Internet Client Error: SSL Error (possibly 32K data upload limitation).
IDS_WinInet_Timeout	8194 -2146820094 0x800A2002	Internet Client Error: Request Timeout.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Internet Information Services Error Codes

The following table lists Microsoft Internet Information Services (IIS) error codes related to Remote Data Service usage. The positive decimal translation of the low two bytes, the negative decimal translation of the full error code, and the hexadecimal values are shown.

Internet Information Services errors	Number	Description
IDS_IIS_AccessDenied	8208 -2146820080 0x800A2010	Internet Server Error: Access Denied.
IDS_IIS_ObjectNotFound	8209 -2146820079 0x800A2011	Internet Server Error: Object/module not found.
IDS_IIS_RequestForbidden	8210 -2146820078 0x800A2012	Internet Server Error: Request Forbidden.
IDS_IIS_UnexpectedError	8447 -2146819841 0x800A20FF	Internet Server Error.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Appendix C: Programming with ADO

ADO is a COM automation interface component that can be used with many programming languages, including Microsoft Visual Basic, VBScript, JScript, Visual C++, and Visual J++. A version of ADO is installed with each of these tools and other applications, such as Microsoft Office and Microsoft SQL Server. The most recent version of ADO is available separately with the Microsoft Data Access Components. See the [Microsoft Data Access Web site](#) for more information.

The library for ADO is msado15.dll and the program ID (ProgID) prefix is "ADODB." For example, to explicitly refer to an ADO [Recordset](#), use ADODB.Recordset.

For more information about programming with ADO in various development environments, see the following topics:

- [Using ADO with Microsoft Visual Basic](#)
- [Using ADO with Scripting Languages](#)
- [Using ADO with Microsoft Visual C++](#)
- [Using ADO with Microsoft Visual J++](#)

See Also

[ADO API Reference](#) | [ADO Samples](#) | [Configuring RDS](#) | [ActiveX Data Objects Start Page](#) | [Appendix A: Providers](#) | [What's New in ADO](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Appendixes

Using ADO with Microsoft Visual Basic

Setting up an ADO project and writing ADO code is similar whether you use Visual Basic or Visual Basic for Applications. This topic addresses using ADO with both Visual Basic and Visual Basic for Applications and notes any differences.

Referencing the ADO Library

The ADO library must be referenced by your project.

To reference ADO from Microsoft Visual Basic

1. In Visual Basic, from the **Project** menu, select **References...**
2. Select **Microsoft ActiveX Data Objects x.x Library** from the list. 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
3. Click **OK**.

You can use ADO just as easily with Visual Basic for Applications, using Microsoft Access, for example.

To reference ADO from Microsoft Access

1. In Microsoft Access, select or create a module from the **Modules** tab in the **Database** window.
2. From the **Tools** menu, select **References...**
3. Select **Microsoft ActiveX Data Objects x.x Library** from the list. Verify that at least the following libraries are also selected:
 - Visual Basic for Applications
 - Microsoft Access 11.0 Object Library (or later)
4. Click **OK**.

Creating ADO Objects in Visual Basic

To create an automation variable and an instance of an object for that variable, you can use two methods: **Dim** or **CreateObject**.

Dim

You can use the **New** keyword with **Dim** to declare and instantiate ADO objects in one step:

```
Dim conn As New ADODB.Connection
```

Alternately, the **Dim** statement declaration and object instantiation can also be two steps:

```
Dim conn As ADODB.Connection  
Set conn = New ADODB.Connection
```

Note It is not required to explicitly use the ADODB progid with the **Dim** statement, assuming you have properly referenced the ADO library in your project. However, using it ensures that you won't have naming conflicts with other libraries.

For example, if you include references to both ADO and DAO in the same project, you should include a qualifier to specify which object model to use when instantiating **Recordset** objects, as in the following code:

```
Dim adoRS As ADODB.Recordset  
Dim daoRS As DAO.Recordset
```

CreateObject

With the **CreateObject** method, the declaration and object instantiation must be two discrete steps:

```
Dim conn1  
Set conn1 = CreateObject("ADODB.Connection") As Object
```

Objects instantiated with **CreateObject** are late-bound, which means that they are not strongly typed and command-line completion is disabled. However, it

does allow you to skip referencing the ADO library from your project, and enables you to instantiate specific versions of objects. For example:

```
Set conn1 = CreateObject("ADODB.Connection.2.0") As Object
```

You could also accomplish this by specifically creating a reference to the ADO version 2.0 type library and creating the object.

Instantiating objects with the **CreateObject** method is typically slower than using the **Dim** statement.

Handling Events

In order to handle ADO events in Microsoft Visual Basic, you must declare a module-level variable using the **WithEvents** keyword. The variable can be declared only as part of a class module and must be declared at the module level. For a more complete discussion of handling ADO events, see [Chapter 7: Handling ADO Events](#).

Visual Basic Examples

Many Visual Basic examples are included with the ADO documentation. For more information, see [ADO Code Examples in Microsoft Visual Basic](#).

See Also

[ActiveX Data Objects Start Page](#) | [Using ADO with Microsoft Visual C++](#) | [Using ADO with Microsoft Visual J++](#) | [Using ADO with Scripting Languages](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Using ADO with Scripting Languages

Within a scripting environment, ADO allows you to expose data by way of server-side scripting. In this scenario, ADO, the underlying OLE DB provider that it utilizes, and any other components needed to reference a given data store are installed on a server running Internet Information Services (IIS). Using Active Server Pages (ASP), ADO is a component referenced in a script that can generate HTML, for example. This HTML content can be passed via HTTP to a client Web browser. Through the use of scripting, the Web page can send actions back to the server-side script, allowing you to update, traverse, or view specific data.

ODBC Data Sources

One notable difference between scripting and non-scripting ADO code is the ODBC Data Source, if used. For non-scripting applications, you can create a User DSN in the ODBC Data Source Administrator. For scripts that are running under IIS, you must create a System DSN; otherwise your scripts won't recognize the data source you created. This applies to any ADO scripting application using the Microsoft OLE DB Provider for ODBC through Microsoft IIS.

Referencing the ADO Library

N/A with scripting languages.

Handling events

N/A with scripting languages.

The following topics contain more specific information about using ADO with scripting languages:

- [ADO in VBScript](#)
- [ADO in JScript](#)

See Also

[ActiveX Data Objects Start Page](#) | [Using ADO with Microsoft Visual Basic](#) | [Using ADO with Microsoft Visual C++](#) | [Using ADO with Microsoft Visual J++](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

VBScript ADO Programming

Creating an ADO Project

Microsoft Visual Basic, Scripting Edition does not support type libraries, so you do not need to reference ADO in your project. Consequently, no associated features such as command line completion are supported. Also, by default, ADO enumerated constants are not defined in VBScript.

However, ADO provides you with two include files containing the following definitions to be used with VBScript:

- For server-side scripting use Adovbs.inc, which is installed in the c:\Program Files\Common Files\System\ado\ folder by default.
- For client-side scripting use Adcvbs.inc, which is installed in the c:\Program Files\Common Files\System\msdac\ folder by default.

You can either copy and paste constant definitions from these files into your ASP pages, or, if you are doing server-side scripting, copy Adovbs.inc file to a folder on your Web site and referencing it from your ASP page like this:

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

Creating ADO Objects in VBScript

You cannot use the **Dim** statement to assign objects to a specific type in VBScript. Also, VBScript does not support the **New** syntax used with the **Dim** statement in Visual Basic for Applications. You must instead use the **CreateObject** function call:

```
Dim Rs1
Set Rs1 = Server.CreateObject( "ADODB.Recordset" )
```

VBScript Examples

The following code is a generic example of VBScript server-side programming in an Active Server Page (ASP) file:

```
<% @LANGUAGE="VBSCRIPT" %>
<% Option Explicit %>
<!--#include File="adovbs.inc"-->
<HTML>
  <BODY BGCOLOR="white" topmargin="10" leftmargin="10">

  <!-- Your ASP Code goes here -->
<%
Dim Source
Dim Connect
Dim Rs1

Source = "SELECT * FROM Authors"
Connect = "Provider=sqloledb;Data Source=srv;" & _
  "Initial Catalog=Pubs;Integrated Security=SSPI;"

Set Rs1 = Server.CreateObject( "ADODB.Recordset" )
Rs1.Open Source, Connect, adOpenForwardOnly
Response.Write("Success!")
%>
  </BODY>
</HTML>
```

More specific VBScript examples are included with the ADO documentation. For more information, see [ADO Code Examples in Microsoft Visual Basic Scripting Edition](#).

Differences Between VBScript and Visual Basic

Using ADO with VBScript is similar to using ADO with Visual Basic in many ways, including how syntax is used. However, some significant differences exist:

- VBScript supports only the Variant data type, which can hold different types of data. You can store the data you need in a Variant data type, and the data will function appropriately due to casting performed by VBScript. It recognizes the type required by ADO, and converts the value in the Variant accordingly.
- You cannot use **on error goto <label>** within VBScript.
- VBScript supports some of the built-in Visual Basic functions such as **Msgbox**, **Date**, and **IsNumeric**. However, because VBScript is a subset of Visual Basic, not all built-in functions are supported. For example, VBScript does not support the **Format** function and the file I/O functions.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

JScript ADO Programming

Creating an ADO Project

Microsoft JScript does not support type libraries, so you do not need to reference ADO in your project. Consequently, no associated features such as command line completion are supported. Also, by default, ADO enumerated constants are not defined in JScript.

However, ADO provides you with two include files containing the following definitions to be used with JScript:

- For server-side scripting use Adojavas.inc, which is installed in the c:\Program Files\Common Files\System\ado\ folder by default.
- For client-side scripting use Adcjavas.inc, which is installed in the c:\Program Files\Common Files\System\msdac\ folder by default.

You can either copy and paste constant definitions from these files into your ASP pages, or, if you are doing server-side scripting, copy Adojavas.inc file to a folder on your Web site and references it from your ASP page like this:

```
<!--#include File="adojavas.inc"-->
```

Creating ADO Objects in JScript

You must instead use the **CreateObject** function call:

```
var Rs1;  
Rs1 = Server.CreateObject("ADODB.Recordset");
```

JScript Example

The following code is a generic example of JScript server-side programming in an Active Server Page (ASP) file that opens a **Recordset** object:

```
<% @LANGUAGE="JScript" %>
<!--#include File="adojavas.inc"-->
<HTML>
<BODY BGCOLOR="white" topmargin="10" leftmargin="10">
<%
var Source = "SELECT * FROM Authors";
var Connect = "Provider=sqloledb;Data Source=srv;" +
    "Initial Catalog=Pubs;Integrated Security=SSPI;"
var Rs1 = Server.CreateObject( "ADODB.Recordset.2.5" );
Rs1.Open(Source,Connect,adOpenForwardOnly);
Response.Write("Success!");
%>
</BODY>
</HTML>
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes 

Using ADO with Microsoft Visual C++

For information about using ADO with Visual C++, see the following sections:

- [Visual C++ ADO Programming](#)
- [Visual C++ Extensions for ADO](#)
- [ADO for Visual C++ Syntax Index for COM](#)
- [ADO for Visual C++ Syntax Index with #import](#)

See Also

[ActiveX Data Objects Start Page](#) | [ADO for Visual C++ Syntax Index for COM](#) | [ADO for Visual C++ Syntax Index with #import](#) | [Using ADO with Microsoft Visual Basic](#) | [Using ADO with Microsoft Visual J++](#) | [Using ADO with Scripting Languages](#) | [Visual C++ Extensions for ADO](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Visual C++ ADO Programming

The ADO API Reference describes the functionality of the ADO application programming interface (API) using a syntax similar to Microsoft Visual Basic. Though the intended audience is all users, ADO programmers employ diverse languages such as Visual Basic, Visual C++ (with and without the **#import** directive), and Visual J++ (with the ADO/WFC class package).

To accommodate this diversity, the [ADO for Visual C++ Syntax Indexes](#) provide Visual C++ language-specific syntax with links to common descriptions of functionality, parameters, exceptional behaviors, and so on, in the API Reference.

ADO is implemented with COM (Component Object Model) interfaces. However, it is easier for programmers to work with COM in certain programming languages than others. For example, nearly all the details of using COM are handled implicitly for Visual Basic programmers, whereas Visual C++ programmers must attend to those details themselves.

The following sections summarize details for C and C++ programmers using ADO and the **#import** directive. It focuses on data types specific to COM (**Variant**, **BSTR**, and **SafeArray**), and error handling (`_com_error`).

Using the #import Compiler Directive

The **#import** Visual C++ compiler directive simplifies working with the ADO methods and properties. The directive takes the name of a file containing a type library, such as the ADO .dll (Msado15.dll), and generates header files containing typedef declarations, smart pointers for interfaces, and enumerated constants. Each interface is encapsulated, or wrapped, in a class.

For each operation within a class (that is, a method or property call), there is a declaration to call the operation directly (that is, the "raw" form of the operation), and a declaration to call the raw operation and throw a COM error if the operation fails to execute successfully. If the operation is a property, there is usually a compiler directive that creates an alternative syntax for the operation that has syntax like Visual Basic.

Operations that retrieve the value of a property have names of the form, **GetProperty**. Operations that set the value of a property have names of the form, **PutProperty**. Operations that set the value of a property with a pointer to an ADO object have names of the form, **PutRefProperty**.

You can get or set a property with calls of these forms:

```
variable = objectPtr->GetProperty(); // get property value
objectPtr->PutProperty(value);       // set property value
objectPtr->PutRefProperty(&value);   // set property with object poi
```

Using Property Directives

The `__declspec(property...)` compiler directive is a Microsoft-specific C language extension that declares a function used as a property to have an alternative syntax. As a result, you can set or get values of a property in a way similar to Visual Basic. For example, you can set and get a property this way:

```
objectPtr->property = value;           // set property value  
variable = objectPtr->property;       // get property value
```

Notice you do not have to code:

```
objectPtr->PutProperty(value);         // set property value  
variable = objectPtr->GetProperty;     // get property value
```

The compiler will generate the appropriate **Get-**, **Put-**, or **PutRefProperty** call based on what alternative syntax is declared and whether the property is being read or written.

The `__declspec(property...)` compiler directive can only declare **get**, **put**, or **get and put** alternative syntax for a function. Read-only operations only have a **get** declaration; write-only operations only have a **put** declaration; operations that are both read and write have both **get** and **put** declarations.

Only two declarations are possible with this directive; however, each property may have three property functions: **GetProperty**, **PutProperty**, and **PutRefProperty**. In that case, only two forms of the property have the alternative syntax.

For example, the **Command** object **ActiveConnection** property is declared with an alternative syntax for **GetActiveConnection** and **PutRefActiveConnection**. The **PutRef-** syntax is a good choice because in practice, you will typically want to put an open **Connection** object (that is, a **Connection** object pointer) in this property. On the other hand, the **Recordset** object has **Get-**, **Put-**, and **PutRefActiveConnection** operations, but no alternative syntax.

Collections, the GetItem Method, and the Item Property

ADO defines several collections, including **Fields**, **Parameters**, **Properties**, and **Errors**. In Visual C++, the **GetItem(index)** method returns a member of the collection. *Index* is a **Variant**, the value of which is either a numerical index of the member in the collection, or a string containing the name of the member.

The **__declspec(property...)** compiler directive declares the **Item** property as an alternative syntax to each collection's fundamental **GetItem()** method. The alternative syntax uses square brackets and looks similar to an array reference. In general, the two forms look like the following:

```
collectionPtr->GetItem(index);  
collectionPtr->Item[index];
```

For example, assign a value to a field of a **Recordset** object, named *rs*, derived from the **authors** table of the **pubs** database. Use the **Item()** property to access the third **Field** of the **Recordset** object **Fields** collection (collections are indexed from zero; assume the third field is named *au_fname*). Then call the **Value()** method on the **Field** object to assign a string value.

This can be expressed in Visual Basic in the following four ways (the last two forms are unique to Visual Basic; other languages do not have equivalents):

```
rs.Fields.Item(2).Value = "value"  
rs.Fields.Item("au_fname").Value = "value"  
rs(2) = "value"  
rs!au_fname = "value"
```

The equivalent in Visual C++ to the first two forms above is:

```
rs->Fields->GetItem(long(2))->PutValue("value");  
rs->Fields->GetItem("au_fname")->PutValue("value");
```

-or- (the alternative syntax for the **Value** property is also shown)

```
rs->Fields->Item[long(2)]->Value = "value";  
rs->Fields->Item["au_fname"]->Value = "value";
```

COM-Specific Data Types

In general, any Visual Basic data type you find in the ADO API Reference has a Visual C++ equivalent. These include standard data types such as **unsigned char** for a Visual Basic **Byte**, **short** for **Integer**, and **long** for **Long**. Look in the Syntax Indexes to see exactly what is required for the operands of a given method or property.

The exceptions to this rule are the data types specific to COM: **Variant**, **BSTR**, and **SafeArray**.

Variant

A **Variant** is a structured data type that contains a value member and a data type member. A **Variant** may contain a wide range of other data types including another Variant, BSTR, Boolean, IDispatch or IUnknown pointer, currency, date, and so on. COM also provides methods that make it easy to convert one data type to another.

The **_variant_t** class encapsulates and manages the **Variant** data type.

When the ADO API Reference says a method or property operand takes a value, it usually means the value is passed in a **_variant_t**.

This rule is explicitly true when the **Parameters** section in the topics of the ADO API Reference says an operand is a **Variant**. One exception is when the documentation explicitly says the operand takes a standard data type, such as **Long** or **Byte**, or an enumeration. Another exception is when the operand takes a **String**.

BSTR

A **BSTR** (**B**asic **STR**ing) is a structured data type that contains a character string and the string's length. COM provides methods to allocate, manipulate, and free a **BSTR**.

The **_bstr_t** class encapsulates and manages the **BSTR** data type.

When the ADO API Reference says a method or property takes a **String** value, it means the value is in the form of a **_bstr_t**.

Casting **_variant_t** and **_bstr_t** Classes

Often it is not necessary to explicitly code a **_variant_t** or **_bstr_t** in an argument to an operation. If the **_variant_t** or **_bstr_t** class has a constructor that matches the data type of the argument, then the compiler will generate the appropriate **_variant_t** or **_bstr_t**.

However, if the argument is ambiguous, that is, the argument's data type matches more than one constructor, you must cast the argument with the appropriate data type to invoke the correct constructor.

For example, the declaration for the **Recordset::Open** method is:

```
HRESULT Open (  
    const _variant_t & Source,  
    const _variant_t & ActiveConnection,  
    enum CursorTypeEnum CursorType,  
    enum LockTypeEnum LockType,  
    long Options );
```

The **ActiveConnection** argument takes a reference to a **_variant_t**, which you may code as a connection string or a pointer to an open **Connection** object.

The correct **_variant_t** will be constructed implicitly if you pass a string such as "DSN=pubs;uid=sa;pwd=", or a pointer such as "(IDispatch *) pConn".

Or you may explicitly code a **_variant_t** containing a pointer such as "**_variant_t**((IDispatch *) pConn, true)". The cast, (IDispatch *), resolves the ambiguity with another constructor that takes a pointer to an IUnknown interface.

It is a crucial, though seldom mentioned fact, that ADO is an IDispatch interface. Whenever a pointer to an ADO object must be passed as a **Variant**, that pointer must be cast as a pointer to an IDispatch interface.

The last case explicitly codes the second boolean argument of the constructor with its optional, default value of true. This argument causes the **Variant** constructor to call its **AddRef()** method, which compensates for ADO

automatically calling the **_variant_t::Release()** method when the ADO method or property call completes.

SafeArray

A **SafeArray** is a structured data type that contains an array of other data types. A **SafeArray** is called *safe* because it contains information about the bounds of each array dimension, and limits access to array elements within those bounds.

When the ADO API Reference says a method or property takes or returns an array, it means the method or property takes or returns a **SafeArray**, not a native C/C++ array.

For example, the second parameter of the **Connection** object **OpenSchema** method requires an array of **Variant** values. Those **Variant** values must be passed as elements of a **SafeArray**, and that **SafeArray** must be set as the value of another **Variant**. It is that other **Variant** that is passed as the second argument of **OpenSchema**.

As further examples, the first argument of the **Find** method is a **Variant** whose value is a one-dimensional **SafeArray**; each of the optional first and second arguments of **AddNew** is a one-dimensional **SafeArray**; and the return value of the **GetRows** method is a **Variant** whose value is a two-dimensional **SafeArray**.

Missing and Default Parameters

Visual Basic allows missing parameters in methods. For example, the **Recordset** object **Open** method has five parameters, but you can skip intermediate parameters and leave off trailing parameters. A default **BSTR** or **Variant** will be substituted depending on the data type of the missing operand.

In C/C++, all operands must be specified. If you want to specify a missing parameter whose data type is a string, specify a **_bstr_t** containing a null string. If you want to specify a missing parameter whose data type is a **Variant**, specify a **_variant_t** with a value of `DISP_E_PARAMNOTFOUND` and a type of `VT_ERROR`. Alternatively, specify the equivalent **_variant_t** constant, **vtMissing**, which is supplied by the **#import** directive.

Three methods are exceptions to the typical use of **vtMissing**. These are the **Execute** methods of the **Connection** and **Command** objects, and the **NextRecordset** method of the **Recordset** object. The following are their signatures:

```
_RecordsetPtr Execute( _bstr_t CommandText, VARIANT * RecordsAffected,  
    long Options ); // Connection  
_RecordsetPtr Execute( VARIANT * RecordsAffected, VARIANT * Parameters,  
    long Options ); // Command  
_RecordsetPtr NextRecordset( VARIANT * RecordsAffected ); // Record
```

The parameters, *RecordsAffected* and *Parameters*, are pointers to a **Variant**. *Parameters* is an input parameter which specifies the address of a **Variant** containing a single parameter, or array of parameters, that will modify the command being executed. *RecordsAffected* is an output parameter that specifies the address of a **Variant**, where the number of rows affected by the method is returned.

In the **Command** object **Execute** method, indicate that no parameters are specified by setting *Parameters* to either **&vtMissing** (which is recommended) or to the null pointer (that is, **NULL** or zero (0)). If *Parameters* is set to the null pointer, the method internally substitutes the equivalent of **vtMissing**, and then completes the operation.

In all the methods, indicate that the number of records affected should not be

returned by setting *RecordsAffected* to the null pointer. In this case, the null pointer is not so much a missing parameter as an indication that the method should discard the number of records affected.

Thus, for these three methods, it is valid to code something such as:

```
pConnection->Execute("commandText", NULL, adCmdText);  
pCommand->Execute(NULL, NULL, adCmdText);  
pRecordset->NextRecordset(NULL);
```

Error Handling

In COM, most operations return an HRESULT return code that indicates whether a function completed successfully. The **#import** directive generates wrapper code around each "raw" method or property and checks the returned HRESULT. If the HRESULT indicates failure, the wrapper code throws a COM error by calling `_com_issue_errorex()` with the HRESULT return code as an argument. COM error objects can be caught in a **try-catch** block. (For efficiency's sake, catch a reference to a **_com_error** object.)

Remember, these are ADO errors: they result from the ADO operation failing. Errors returned by the underlying provider appear as **Error** objects in the **Connection** object **Errors** collection.

The **#import** directive creates only error handling routines for methods and properties declared in the ADO .dll. However, you can take advantage of this same error handling mechanism by writing your own error checking macro or inline function. See the topic, [Visual C++ Extensions](#), or the code in the following sections for examples.

Visual C++ Equivalents of Visual Basic Conventions

The following is a summary of several conventions in the ADO documentation, coded in Visual Basic, as well as their equivalents in Visual C++.

Declaring an ADO Object

In Visual Basic, an ADO object variable (in this case for a **Recordset** object) is declared as follows:

```
Dim rst As ADODB.Recordset
```

The clause, "ADODB.Recordset", is the ProgID of the **Recordset** object as defined in the Registry. A new instance of a **Record** object is declared as follows:

```
Dim rst As New ADODB.Recordset
```

-or-

```
Dim rst As ADODB.Recordset  
Set rst = New ADODB.Recordset
```

In Visual C++, the **#import** directive generates smart pointer-type declarations for all the ADO objects. For example, a variable that points to a **_Recordset** object is of type **_RecordsetPtr**, and is declared as follows:

```
_RecordsetPtr rs;
```

A variable that points to a new instance of a **_Recordset** object is declared as follows:

```
_RecordsetPtr rs("ADODB.Recordset");
```

-or-

```
_RecordsetPtr rs;  
rs.CreateInstance("ADODB.Recordset");
```

-or-

```
_RecordsetPtr rs;  
rs.CreateInstance(__uuidof(_Recordset));
```

After the **CreateInstance** method is called, the variable can be used as follows:

```
rs->Open(...);
```

Notice that in one case, the "." operator is used as if the variable were an instance of a class (`rs.CreateInstance`), and in another case, the "->" operator is used as if the variable were a pointer to an interface (`rs->Open`).

One variable can be used in two ways because the "->" operator is overloaded to allow an instance of a class to behave like a pointer to an interface. A private class member of the instance variable contains a pointer to the **_Recordset** interface; the "->" operator returns that pointer; and the returned pointer accesses the members of the **_Recordset** object.

Coding a Missing Parameter — String

When you need to code a missing **String** operand in Visual Basic, you merely omit the operand. You must specify the operand in Visual C++. Code a **_bstr_t** that has an empty string as a value.

```
_bstr_t strMissing(L "");
```

Coding a Missing Parameter — Variant

When you need to code a missing **Variant** operand in Visual Basic, you merely omit the operand. You must specify all operands in Visual C++. Code a missing **Variant** parameter with a **_variant_t** set to the special value, `DISP_E_PARAMNOTFOUND`, and type, `VT_ERROR`. Alternatively, specify **vtMissing**, which is an equivalent pre-defined constant supplied by the **#import** directive.

```
_variant_t vtMissingYours(DISP_E_PARAMNOTFOUND, VT_ERROR);
```

-or use -

```
...vtMissing...;
```

Declaring a Variant

In Visual Basic, a **Variant** is declared with the **Dim** statement as follows:

```
Dim VariableName As Variant
```

In Visual C++, declare a variable as type **_variant_t**. A few schematic **_variant_t** declarations are shown below.

Note These declarations merely give a rough idea of what you would code in your own program. For more information, see the examples below, and the Visual C++ documentation.

```
_variant_t VariableName(value);  
_variant_t VariableName((data type cast) value);  
_variant_t VariableName(value, VT_DATATYPE);  
_variant_t VariableName(interface * value, bool fAddRef = true);
```

Using Arrays of Variants

In Visual Basic, arrays of **Variants** can be coded with the **Dim** statement, or you may use the **Array** function, as demonstrated in the following example code:

```
Public Sub ArrayOfVariants  
Dim cn As ADODB.Connection  
Dim rs As ADODB.Recordset  
Dim fld As ADODB.Field  
  
cn.Open "DSN=pubs", "sa", ""  
rs = cn.OpenSchema(adSchemaColumns, _  
                  Array(Empty, Empty, "authors", Empty))  
For Each fld in rs.Fields  
    Debug.Print "Name = "; fld.Name  
Next fld  
rs.Close  
cn.Close  
End Sub
```

The following Visual C++ example demonstrates using a **SafeArray** used with a **_variant_t**.

Notes The following notes correspond to commented sections in the code example.

1. Once again, the TESTHR() inline function is defined to take advantage of the existing error-handling mechanism.

2. You only need a one-dimensional array, so you can use **SafeArrayCreateVector**, instead of the general purpose **SAFEARRAYBOUND** declaration and **SafeArrayCreate** function. The following is what that code would look like using **SafeArrayCreate**:

```
SAFEARRAYBOUND  sabound[1];
    sabound[0].lLbound = 0;
    sabound[0].cElements = 4;
    pSa = SafeArrayCreate(VT_VARIANT, 1, sabound);
```

3. The schema identified by the enumerated constant, **adSchemaColumns**, is associated with four constraint columns: **TABLE_CATALOG**, **TABLE_SCHEMA**, **TABLE_NAME**, and **COLUMN_NAME**. Therefore, an array of **Variant** values with four elements is created. Then a constraint value that corresponds to the third column, **TABLE_NAME**, is specified.

The **Recordset** that is returned consists of several columns, a subset of which is the constraint columns. The values of the constraint columns for each returned row must be the same as the corresponding constraint values.

4. Those familiar with **SafeArrays** may be surprised that **SafeArrayDestroy()** is not called before the exit. In fact, calling **SafeArrayDestroy()** in this case will cause a run-time exception. The reason is that the destructor for **vtCriteria** will call **VariantClear()** when the **_variant_t** goes out of scope, which will free the **SafeArray**. Calling **SafeArrayDestroy**, without manually clearing the **_variant_t**, would cause the destructor to try to clear an invalid **SafeArray** pointer.

If **SafeArrayDestroy** were called, the code would look like this:

```
    TESTHR(SafeArrayDestroy(pSa));
    vtCriteria.vt = VT_EMPTY;
    vtCriteria.parray = NULL;
```

However, it is much simpler to let the **_variant_t** manage the **SafeArray**.

```
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")
#include <stdio.h>

// Note 1
inline void TESTHR( HRESULT _hr )
    { if FAILED(_hr) _com_issue_error(_hr); }
```

```

void main(void)
{
    CoInitialize(NULL);
    try
    {
        _RecordsetPtr    pRs("ADODB.Recordset");
        _ConnectionPtr  pCn("ADODB.Connection");
        _variant_t      vtTableName("authors"),
                       vtCriteria;
        long            ix[1];
        SAFEARRAY       *pSa = NULL;

        pCn->Open("DSN=pubs;User ID=MyUserId;pwd=MyPassword;Provider=MSDA
                adConnectUnspecified);
// Note 2, Note 3
        pSa = SafeArrayCreateVector(VT_VARIANT, 1, 4);
        if (!pSa) _com_issue_error(E_OUTOFMEMORY);

// Specify TABLE_NAME in the third array element (index of 2).

        ix[0] = 2;
        TESTHR(SafeArrayPutElement(pSa, ix, &vtTableName));

//   There is no Variant constructor for a SafeArray, so manually se
//   type (SafeArray of Variant) and value (pointer to a SafeArray).

        vtCriteria.vt = VT_ARRAY | VT_VARIANT;
        vtCriteria.parray = pSa;

        pRs = pCn->OpenSchema(adSchemaColumns, vtCriteria, vtMissing);

        long limit = pRs->GetFields()->Count;
        for (long x = 0; x < limit; x++)
            printf("%d: %s\n", x+1,
                ((char*) pRs->GetFields()->Item[x]->Name));
// Note 4
        pRs->Close();
        pCn->Close();
    }
    catch (_com_error &e)
    {
        printf("Error:\n");
        printf("Code = %08lx\n", e.Error());
        printf("Code meaning = %s\n", (char*) e.ErrorMessage());
        printf("Source = %s\n", (char*) e.Source());
        printf("Description = %s\n", (char*) e.Description());
    }
    CoUninitialize();
}

```

Using Property Get/Put/PutRef

In Visual Basic, the name of a property is not qualified by whether it is retrieved, assigned, or assigned a reference.

```
Public Sub GetPutPutRef
Dim rs As New ADODB.Recordset
Dim cn As New ADODB.Connection
Dim sz as Integer
cn.Open "Provider=sqloledb;Data Source=yourserver;" & _
        "Initial Catalog=pubs;Integrated Security=SSPI;"
rs.PageSize = 10
sz = rs.PageSize
rs.ActiveConnection = cn
rs.Open "authors",,adOpenStatic
' ...
rs.Close
cn.Close
End Sub
```

This Visual C++ example demonstrates the **Get/Put/PutRefProperty**.

Notes The following notes correspond to commented sections in the code example.

1. This example uses two forms of a missing string argument: an explicit constant, **strMissing**, and a string that the compiler will use to create a temporary **_bstr_t** that will exist for the scope of the **Open** method.

2. It isn't necessary to cast the operand of `rs->PutRefActiveConnection(cn)` to `(IDispatch *)` because the type of the operand is already `(IDispatch *)`.

```
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
        no_namespace rename("EOF", "EndOfFile")
#include <stdio.h>

void main(void)
{
    CoInitialize(NULL);
    try
    {
        _ConnectionPtr  cn("ADODB.Connection");
        _RecordsetPtr  rs("ADODB.Recordset");
```

```

        _bstr_t          strMissing(L"");
        long            oldPgSz = 0,
                       newPgSz = 5;

// Note 1
    cn->Open("Provider=sqloledb;Data Source=MyServer;"
           "Initial Catalog=pubs;Integrated Security=SSPI;",
           strMissing, "",
           adConnectUnspecified);

    oldPgSz = rs->GetPageSize();
// -or-
    oldPgSz = rs->PageSize;

    rs->PutPageSize(newPgSz);
// -or-
    rs->PageSize = newPgSz;

// Note 2
    rs->PutRefActiveConnection( cn );
    rs->Open("authors", vtMissing, adOpenStatic, adLockReadOnly,
           adCmdTable);
    printf("Original pagesize = %d, new pagesize = %d\n", oldPgSz,
           rs->GetPageSize());
    rs->Close();
    cn->Close();
}
catch (_com_error &e)
{
    printf("Description = %s\n", (char*) e.Description());
}
::CoUninitialize();
}

```

Using GetItem(x) and Item[x]

This Visual Basic example demonstrates the standard and alternative syntax for **Item()**.

```

Public Sub GetItemItem
Dim rs As New ADODB.Recordset
Dim name as String
rs = rs.Open "authors", "DSN=pubs;", adOpenDynamic, _
           adLockBatchOptimistic, adTable
name = rs(0)
' -or-
name = rs.Fields.Item(0)
rs(0) = "Test"

```

```

rs.UpdateBatch
' Restore name
rs(0) = name
rs.UpdateBatch
rs.Close
End Sub

```

This Visual C++ example demonstrates **Item**.

Note The following note corresponds to commented sections in the code example.

1. When the collection is accessed with **Item**, the index, **2**, must be cast to **long** so an appropriate constructor will be invoked.

```

#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")
#include <stdio.h>

void main(void)
{
    CoInitialize(NULL);
    try {
        _RecordsetPtr    rs("ADODB.Recordset");
        _variant_t       vtFirstName;

        rs->Open("authors",
                "Provider=sqloledb;Data Source=MyServer;"
                "Initial Catalog=pubs;Integrated Security=SSPI;",
                adOpenStatic, adLockOptimistic, adCmdTable);
        rs->MoveFirst();

// Note 1. Get a field.
        vtFirstName = rs->Fields->GetItem((long)2)->GetValue();
// -or-
        vtFirstName = rs->Fields->Item[(long)2]->Value;

        printf( "First name = '%s'\n", (char*) ((_bstr_t) vtFirstName)

        rs->Fields->GetItem((long)2)->Value = L"TEST";
        rs->Update(vtMissing, vtMissing);

// Restore name
        rs->Fields->GetItem((long)2)->PutValue(vtFirstName);
// -or-
        rs->Fields->GetItem((long)2)->Value = vtFirstName;
        rs->Update(vtMissing, vtMissing);
        rs->Close();
    }
}

```

```

    }
    catch (_com_error &e)
    {
        printf("Description = '%s'\n", (char*) e.Description());
    }
    ::CoUninitialize();
}

```

Casting ADO object pointers with (IDispatch *)

The following Visual C++ example demonstrates using (IDispatch *) to cast ADO object pointers.

Notes The following notes correspond to commented sections in the code example.

1. Specify an open **Connection** object in an explicitly coded **Variant**. Cast it with (IDispatch *) so the correct constructor will be invoked. Also, explicitly set the second **_variant_t** parameter to the default value of **true**, so the object reference count will be correct when the **Recordset::Open** operation ends.

2. The expression, (**_bstr_t**), is not a cast, but a **_variant_t** operator that extracts a **_bstr_t** string from the **Variant** returned by **Value**.

The expression, (**char***), is not a cast, but a **_bstr_t** operator that extracts a pointer to the encapsulated string in a **_bstr_t** object.

This section of code demonstrates some of the useful behaviors of **_variant_t** and **_bstr_t** operators.

```

#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>

void main(void)
{
    CoInitialize(NULL);
    try
    {
        _ConnectionPtr pConn("ADODB.Connection");
        _RecordsetPtr pRst("ADODB.Recordset");
    }
}

```

```

        pConn->Open("Provider=sqloledb;Data Source=MyServer;"
            "Initial Catalog=pubs;Integrated Security=SSPI;",
            "", "", adConnectUnspecified);
// Note 1.
    pRst->Open(
        "authors",
        _variant_t((IDispatch *) pConn, true),
        adOpenStatic,
        adLockReadOnly,
        adCmdTable);
    pRst->MoveLast();
// Note 2.
    printf("Last name is '%s %s'\n",
        (char*) ((_bstr_t) pRst->GetFields()->GetItem("au_fname")
        (char*) ((_bstr_t) pRst->Fields->Item["au_lname"]->Value

        pRst->Close();
        pConn->Close();
    }
    catch (_com_error &e)
    {
        printf("Description = '%s'\n", (char*) e.Description());
    }
    ::CoUninitialize();
}

```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Visual C++ Extensions for ADO

The preferred method of programming ADO with Visual C++ is using the **#import** directive, as discussed in [Microsoft Visual C++ ADO Programming](#). However, earlier versions of ADO shipped with an alternate method of programming using Visual C++: the Visual C++ Extensions. This section documents this feature for those who must maintain Visual C++ Extensions code, but new ADO code should be written using **#import**.

One of the most tedious jobs Visual C++ programmers face when retrieving data with ADO is converting data returned as a VARIANT data type into a C++ data type, and then storing the converted data in a class or structure. In addition to being cumbersome, retrieving C++ data through a VARIANT data type diminishes performance.

ADO provides an interface that supports retrieving data into native C/C++ data types without going through a VARIANT, and also provides preprocessor macros that simplify using the interface. The result is a flexible tool that is easier to use and has great performance.

A common C/C++ client scenario is to bind a record in a [Recordset](#) to a C/C++ struct or class containing native C/C++ types. When going through VARIANTs, this involves writing conversion code from VARIANT to C/C++ native types. The Visual C++ Extensions for ADO are targeted at making this scenario much easier for the Visual C++ programmer.

See the following topics to learn more about the Visual C++ Extensions for ADO.

- [Using Visual C++ Extensions for ADO](#)
- [Visual C++ Extensions Header](#)
- [ADO with Visual C++ Extensions Example](#)

See Also

[ADO for Visual C++ Syntax Index for COM](#) | [ADO with Visual C++ Extensions Example](#) | [Using Visual C++ Extensions for ADO](#) | [Visual C++ Extensions](#)

Header

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Appendixes

Using Visual C++ Extensions

The IADORecordBinding Interface

The Microsoft Visual C++ Extensions for ADO associate, or bind, fields of a [Recordset](#) object to C/C++ variables. Whenever the current row of the bound **Recordset** changes, all the bound fields in the **Recordset** are copied to the C/C++ variables. If necessary, the copied data is converted to the declared data type of the C/C++ variable.

The **BindToRecordset** method of the **IADORecordBinding** interface binds fields to C/C++ variables. The **AddNew** method adds a new row to the bound **Recordset**. The **Update** method populates fields in new rows of the **Recordset**, or updates fields in existing rows, with the value of the C/C++ variables.

The **IADORecordBinding** interface is implemented by the **Recordset** object. You do not code the implementation yourself.

Binding Entries

The Visual C++ Extensions for ADO map fields of a [Recordset](#) object to C/C++ variables. The definition of a mapping between a field and a variable is called a *binding entry*. Macros provide binding entries for numeric, fixed-length, and variable-length data. The binding entries and C/C++ variables are declared in a class derived from the Visual C++ Extensions class, **CADORecordBinding**. The **CADORecordBinding** class is defined internally by the binding entry macros.

ADO internally maps the parameters in these macros to an OLE DB **DBBINDING** structure and creates an OLE DB **Accessor** object to manage the movement and conversion of data between fields and variables. OLE DB defines data as consisting of three parts: A *buffer* where the data is stored; a *status* that indicates whether a field was successfully stored in the buffer, or how the variable should be restored to the field; and the *length* of the data. (See the *OLE DB Programmer's Reference*, Chapter 6: Getting and Setting Data for more information.)

Header File

Include the following file in your application in order to use the Visual C++ Extensions for ADO:

```
#include <icrsint.h>
```

Binding Recordset Fields

To Bind Recordset Fields to C/C++ Variables

1. Create a class derived from the **CADORecordBinding** class.
2. Specify binding entries and corresponding C/C++ variables in the derived class. Bracket the binding entries between **BEGIN_ADO_BINDING** and **END_ADO_BINDING** macros. Do not terminate the macros with commas or semicolons. Appropriate delimiters are specified automatically by each macro.

Specify one binding entry for each field to be mapped to a C/C++ variable. Use an appropriate member from the **ADO_FIXED_LENGTH_ENTRY**, **ADO_NUMERIC_ENTRY**, or **ADO_VARIABLE_LENGTH_ENTRY** family of macros.

3. In your application, create an instance of the class derived from **CADORecordBinding**. Get the **IADORecordBinding** interface from the **Recordset**. Then call the **BindToRecordset** method to bind the **Recordset** fields to the C/C++ variables.

[See the Visual C++ Extensions Example](#) for more information.

Interface Methods

The **IADORecordBinding** interface has three methods: **BindToRecordset**, **AddNew**, and **Update**. The sole argument to each method is a pointer to an instance of the class derived from **CADORecordBinding**. Therefore, the **AddNew** and **Update** methods cannot specify any of the parameters of their ADO method namesakes.

Syntax

The **BindToRecordset** method associates the **Recordset** fields with C/C++ variables.

```
BindToRecordset(CADORecordBinding *binding)
```

The **AddNew** method invokes its namesake, the ADO [AddNew](#) method, to add a new row to the **Recordset**.

```
AddNew(CADORecordBinding *binding)
```

The **Update** method invokes its namesake, the ADO [Update](#) method, to update the **Recordset**.

```
Update(CADORecordBinding *binding)
```

Binding Entry Macros

Binding entry macros define the association of a **Recordset** field and a variable. A beginning and ending macro delimits the set of binding entries.

Families of macros are provided for fixed-length data, such as **adDate** or **adBoolean**; numeric data, such as **adTinyInt**, **adInteger**, or **adDouble**; and variable-length data, such as **adChar**, **adVarChar** or **adVarBinary**. All numeric types, except for **adVarNumeric**, are also fixed-length types. Each family has differing sets of parameters so that you can exclude binding information that is of no interest.

See the *OLE DB Programmer's Reference*, Appendix A: Data Types for additional information.

Begin Binding Entries

BEGIN_ADO_BINDING(*Class*)

Fixed-Length Data

ADO_FIXED_LENGTH_ENTRY(*Ordinal, DataType, Buffer, Status, Modify*)

ADO_FIXED_LENGTH_ENTRY2(*Ordinal, DataType, Buffer, Modify*)

Numeric Data

ADO_NUMERIC_ENTRY(*Ordinal, DataType, Buffer, Precision, Scale, Status, Modify*)

ADO_NUMERIC_ENTRY2(*Ordinal, DataType, Buffer, Precision, Scale, Modify*)

Variable-Length Data

ADO_VARIABLE_LENGTH_ENTRY(*Ordinal, DataType, Buffer, Size, Status,*

Length, Modify)

ADO_VARIABLE_LENGTH_ENTRY2(*Ordinal, DataType, Buffer, Size, Status,*

Modify)

ADO_VARIABLE_LENGTH_ENTRY3(*Ordinal, DataType, Buffer, Size, Length,*

Modify)

ADO_VARIABLE_LENGTH_ENTRY4(*Ordinal, DataType, Buffer, Size, Modify)*

End Binding Entries

END_ADO_BINDING()

Parameter	Description
<i>Class</i>	Class in which the binding entries and C/C++ variables are defined.
<i>Ordinal</i>	Ordinal number, counting from one, of the Recordset field corresponding to your C/C++ variable.
<i>DataType</i>	Equivalent ADO data type of the C/C++ variable (see DataTypeEnum for a list of valid data types). The value of the Recordset field will be converted to this data type if necessary.
<i>Buffer</i>	Name of the C/C++ variable where the Recordset field will be stored.
<i>Size</i>	Maximum size in bytes of <i>Buffer</i> . If <i>Buffer</i> will contain a variable-length string, allow room for a terminating zero. Name of a variable that will indicate whether the contents of <i>Buffer</i> are valid, and whether the conversion of the field to <i>DataType</i> was successful.
<i>Status</i>	The two most important values for this variable are adFldOK , which means the conversion was successful; and adFldNull , which means the value of the field would be a VARIANT of type VT_NULL and not merely empty. Possible values for <i>Status</i> are listed in the next table, "Status Values." Boolean flag; if TRUE, indicates ADO is allowed to update the corresponding Recordset field with the value contained

in *Buffer*.

Modify

Set the Boolean *modify* parameter to TRUE to enable ADO to update the bound field, and FALSE if you want to examine the field but not change it.

Precision

Number of digits that can be represented in a numeric variable.

Scale

Number of decimal places in a numeric variable.

Length

Name of a four-byte variable that will contain the actual length of the data in *Buffer*.

Status Values

The value of the *Status* variable indicates whether a field was successfully copied to a variable.

When setting data, *Status* may be set to **adFldNull** to indicate the **Recordset** field should be set to null.

Constant	Value	Description
adFldOK	0	A non-null field value was returned.
adFldBadAccessor	1	Binding was invalid.
adFldCantConvertValue	2	Value couldn't be converted for reasons other than sign mismatch or data overflow.
adFldNull	3	When getting a field, indicates a null value was returned. When setting a field, indicates the field should be set to NULL when the field cannot encode NULL itself (for example, a character array or an integer).
adFldTruncated	4	Variable-length data or numeric digits were truncated.
adFldSignMismatch	5	Value is signed and variable data type is unsigned.
adFldDataOverFlow	6	Value is larger than could be stored in the variable data type.
adFldCantCreate	7	Unknown column type and field already open.
adFldUnavailable	8	Field value could not be determined—for example, on a new, unassigned field with no default value.
adFldPermissionDenied	9	When updating, no permission to write data.

adFldIntegrityViolation	10	When updating, field value would violate column integrity.
adFldSchemaViolation	11	When updating, field value would violate column schema.
adFldBadStatus	12	When updating, invalid status parameter.
adFldDefault	13	When updating, a default value was used.

See Also

[ADO with Visual C++ Extensions Example](#) | [Visual C++ Extensions Header](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Visual C++ Extensions Header

The following header, **icrsint.h**, details the interface that allow clients to retrieve fields from a **Recordset** into variables defined in a class derived from **CADORecordBinding**. You must specify an ADO binding macro for each field you intend to access.

```
#ifndef _ICRSINT_H_
#define _ICRSINT_H_

#include <olectl.h>
#include <stddef.h>

// forwards
class CADORecordBinding;

#define classoffset(base, derived) ((DWORD)(static_cast<base*>((deri

enum ADOFieldStatusEnum
{
    adFldOK = 0,
    adFldBadAccessor = 1,
    adFldCantConvertValue = 2,
    adFldNull = 3,
    adFldTruncated = 4,
    adFldSignMismatch = 5,
    adFldDataOverflow = 6,
    adFldCantCreate = 7,
    adFldUnavailable = 8,
    adFldPermissionDenied = 9,
    adFldIntegrityViolation = 10,
    adFldSchemaViolation = 11,
    adFldBadStatus = 12,
    adFldDefault = 13
};

typedef struct stADO_BINDING_ENTRY
{
    ULONG        ulOrdinal;
    WORD         wDataType;
    BYTE         bPrecision;
    BYTE         bScale;
    ULONG        ulSize;
    ULONG        ulBufferOffset;
    ULONG        ulStatusOffset;
```

```

        ULONG        ulLengthOffset;
        ULONG        ulADORecordBindingOffset;
        BOOL         fModify;
    } ADO_BINDING_ENTRY;

#define BEGIN_ADO_BINDING(cls) public: \
    typedef cls ADORowClass; \
    const ADO_BINDING_ENTRY* STDMETHODCALLTYPE GetADOBindingEntries()
    static const ADO_BINDING_ENTRY rgADOBindingEntries[] = {

//
// Fixed length non-numeric data
//
#define ADO_FIXED_LENGTH_ENTRY(Ordinal, DataType, Buffer, Status, Mo
    {Ordinal, \
    DataType, \
    0, \
    0, \
    0, \
    offsetof(ADORowClass, Buffer), \
    offsetof(ADORowClass, Status), \
    0, \
    classoffset(CADORecordBinding, ADORowClass), \
    Modify},

#define ADO_FIXED_LENGTH_ENTRY2(Ordinal, DataType, Buffer, Modify)\
    {Ordinal, \
    DataType, \
    0, \
    0, \
    0, \
    offsetof(ADORowClass, Buffer), \
    0, \
    0, \
    classoffset(CADORecordBinding, ADORowClass), \
    Modify},

//
// Numeric data
//
#define ADO_NUMERIC_ENTRY(Ordinal, DataType, Buffer, Precision, Scal
    {Ordinal, \
    DataType, \
    Precision, \
    Scale, \
    0, \
    offsetof(ADORowClass, Buffer), \
    offsetof(ADORowClass, Status), \
    0, \
    classoffset(CADORecordBinding, ADORowClass), \

```

```

    Modify},

#define ADO_NUMERIC_ENTRY2(Ordinal, DataType, Buffer, Precision, Scale,
    {Ordinal, \
    DataType, \
    Precision, \
    Scale, \
    0, \
    offsetof(ADORowClass, Buffer), \
    0, \
    0, \
    classoffset(CADORecordBinding, ADORowClass), \
    Modify},

//
// Variable length data
//
#define ADO_VARIABLE_LENGTH_ENTRY(Ordinal, DataType, Buffer, Size, Status,
    {Ordinal, \
    DataType, \
    0, \
    0, \
    Size, \
    offsetof(ADORowClass, Buffer), \
    offsetof(ADORowClass, Status), \
    offsetof(ADORowClass, Length), \
    classoffset(CADORecordBinding, ADORowClass), \
    Modify},

#define ADO_VARIABLE_LENGTH_ENTRY2(Ordinal, DataType, Buffer, Size, Status,
    {Ordinal, \
    DataType, \
    0, \
    0, \
    Size, \
    offsetof(ADORowClass, Buffer), \
    offsetof(ADORowClass, Status), \
    0, \
    classoffset(CADORecordBinding, ADORowClass), \
    Modify},

#define ADO_VARIABLE_LENGTH_ENTRY3(Ordinal, DataType, Buffer, Size, Status,
    {Ordinal, \
    DataType, \
    0, \
    0, \
    Size, \
    offsetof(ADORowClass, Buffer), \
    0, \

```

```

        offsetof(ADORowClass, Length), \
        classoffset(CADORecordBinding, ADORowClass), \
        Modify},

#define ADO_VARIABLE_LENGTH_ENTRY4(Ordinal, DataType, Buffer, Size,
    {Ordinal, \
    DataType, \
    0, \
    0, \
    Size, \
    offsetof(ADORowClass, Buffer), \
    0, \
    0, \
    classoffset(CADORecordBinding, ADORowClass), \
    Modify},

#define END_ADO_BINDING()    {0, adEmpty, 0, 0, 0, 0, 0, 0, 0, FALSE}
    return rgADOBindingEntries;}

//
// Interface that the client 'record' class needs to support. The AD
// provide the implementation for this interface.
//
class CADORecordBinding
{
public:
    STDMETHODCALLTYPE_(const ADO_BINDING_ENTRY*, GetADOBindingEntries) (VOID)
};

//
// Interface that allows a client to fetch a record of data into cla
//
struct __declspec(uuid("00000544-0000-0010-8000-00aa006d2ea4")) IADO
DECLARE_INTERFACE_(IADORecordBinding, IUnknown)
{
public:
    STDMETHODCALLTYPE_(BindToRecordset) (CADORecordBinding *pAdoRecordBinding)
    STDMETHODCALLTYPE_(AddNew) (CADORecordBinding *pAdoRecordBinding) PURE;
    STDMETHODCALLTYPE_(Update) (CADORecordBinding *pAdoRecordBinding) PURE;
};

#endif // !_ICRSINT_H_

```

See Also

[ADO with Visual C++ Extensions Example](#) | [Using Visual C++ Extensions for ADO](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Appendixes

Visual C++ Extensions Example

This program shows how values are retrieved from fields and converted to C/C++ variables.

This example also takes advantage of "smart pointers," which automatically handle the COM-specific details of calling `QueryInterface` and reference counting for the **IADORecordBinding** interface.

Without smart pointers, you would code:

```
IADORecordBinding *picRs = NULL;
...
TESTHR(pRs->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));
...
if (picRs) picRs->Release();
```

With smart pointers, you derive the `IADORecordBindingPtr` type from the `IADORecordBinding` interface with this statement:

```
_COM_SMARTPTR_TYPEDEF(IADORecordBinding, __uuidof(IADORecordBinding))
```

And instantiate the pointer like this:

```
IADORecordBindingPtr picRs(pRs);
```

Because the Visual C++ Extensions are implemented by the **Recordset** object, the constructor for the smart pointer, `picRs`, takes the `_RecordsetPtr` pointer, `pRs`. The constructor calls `QueryInterface` using `pRs` to find the `IADORecordBinding` interface.

```
// Visual C++ Extensions Example
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <icrsint.h>
_COM_SMARTPTR_TYPEDEF(IADORecordBinding, __uuidof(IADORecordBinding))

inline void TESTHR(HRESULT _hr) { if FAILED(_hr) _com_issue_error(_h
```

```

class CCustomRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CCustomRs)
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_ch_fname,
                                sizeof(m_ch_fname), m_ul_fnameStatus, false)
    ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_ch_lname,
                                sizeof(m_ch_lname), m_ul_lnameStatus, false)
END_ADO_BINDING()
public:
    CHAR    m_ch_fname[22];
    CHAR    m_ch_lname[32];
    ULONG   m_ul_fnameStatus;
    ULONG   m_ul_lnameStatus;
};

void main(void)
{
    ::CoInitialize(NULL);
    try
    {
        _RecordsetPtr pRs("ADODB.Recordset");
        CCustomRs rs;
        IADORecordBindingPtr picRs(pRs);

        pRs->Open("SELECT * FROM Employee ORDER BY lname",
                 "dsn=pubs;uid=sa;pwd=",
                 adOpenStatic, adLockOptimistic, adCmdText);

        TESTHR(picRs->BindToRecordset(&rs));

        while (!pRs->EndOfFile)
        {
            // Process data in the CCustomRs C++ instance variables.
            printf("Name = %s %s\n",
                  (rs.m_ul_fnameStatus == adFldOK ? rs.m_ch_fname: "<Error
                  (rs.m_ul_lnameStatus == adFldOK ? rs.m_ch_lname: "<Error

            // Move to the next row of the Recordset.
            // Fields in the new row will automatically be
            // placed in the CCustomRs C++ instance variables.

            pRs->MoveNext();
        }
    }
    catch (_com_error &e )
    {
        printf("Error:\n");
        printf("Code = %08lx\n", e.Error());
        printf("Meaning = %s\n", e.ErrorMessage());
    }
}

```

```
        printf("Source = %s\n", (LPCSTR) e.Source());  
        printf("Description = %s\n", (LPCSTR) e.Description());  
    }  
    ::CoUninitialize();  
}
```

See Also

[Using Visual C++ Extensions for ADO](#) | [Visual C++ Extensions Header](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Using ADO with Microsoft Visual J++

You can implement ADO using Visual J++ in the following ways:

- With Visual J++ 6.0 (or later), use [ADO for Windows Foundation Classes](#).
- With Visual J++ 1.x, use the [Java Type Library Wizard](#).
- Use the [Microsoft SDK for Java](#) utilities.

For further information about using ADO with Visual J++, see the following topics:

- [ADO Java Class Wrappers](#)
- [ADO/WFC Syntax Index](#)

See Also

[ADO for Windows Foundation Classes](#) | [ADO Java Class Wrappers](#) | [ADO/WFC Syntax Index](#) | [Using ADO with Microsoft Visual Basic](#) | [Using ADO with Microsoft Visual C++](#) | [Using ADO with Scripting Languages](#) | [Using ADO with the Java Type Library Wizard](#) | [Using ADO with the Microsoft SDK for Java](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

ADO/WFC Programming

For Microsoft Visual J++ 6.0, ADO has been extended to work with Windows Foundation Classes (WFC) in the following ways. First, a set of Java classes has been implemented that extends the ADO interfaces and creates notifications interesting to the Java programmer; the Java classes also expose functions that return Java types to the user. To improve performance, the Java class directly accesses the native data types in the OLE DB rowset object, and returns them to the Java programmer as Java types without first converting them to and from a variant. ADO has also been extended to work with event notifications in the WFC framework.

ADO for Windows Foundation Classes (ADO/WFC) supports all the standard ADO methods, properties, objects, and events. However, operations that require a variant as a parameter and show excellent performance in a language such as Microsoft Visual Basic, display lesser performance in a language such as Visual J++. For that reason, ADO/WFC also provides accessor functions on the [Field](#) object that take native Java data types instead of the variant data type.

For more detailed information within the ADO documentation about ADO/WFC packages, see [the ADO/WFC Syntax Index](#).

For related information within the Visual J++ documentation, see [Converting Visual Basic ADO Examples to Visual J++](#).

Referencing the Library

To import the ADO/WFC data classes into your project, include the following line in your code:

```
import com.ms.wfc.data.*;
```

See Also

[ADO Event Instantiation by Language](#) | [ADO/WFC Syntax Index](#) | [Using ADO with the Java Type Library Wizard](#) | [Using ADO with the Microsoft SDK for Java](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Using the Java Type Library Wizard

The Java Type Library Wizard is a feature of Visual J++ 1.x, integrated into the **Tools** menu of the development environment. Its purpose is to search a type library and create a Java interface that allows access to COM objects. For Visual J++ 6.0, the Java Type Library Wizard has been replaced with [ADO for Windows Foundation Classes](#).

The Java Type Library Wizard produces similar results as the command-line tools included with the [Microsoft SDK for Java](#). However, you cannot step into the class wrappers that the Wizard generates, unlike the class wrappers generated by the Microsoft SDK for Java.

The Java Type Library Wizard generates the classes in the following location: \<windows directory>\Java\trustlib\msado15. The Summary.txt file, located in the directory where the classes were generated, shows the class definitions it generated.

The Java Type Library Wizard converts enumerated types, found in any given type library, to type INT (integer). It also defines an interface that corresponds to each enumerated type in the type library. You can reference the values of an ADO enumerated type with the following syntax:

```
msado15.<Enum Name>.<constant Name>
```

An example of this is shown in the following code fragment for setting the **CommandType** property of a **Command** object:

```
Cmd1.putCommandType( msado15.CommandTypeEnum.adCmdStoredProc );
```

Alternately, you could inherit from the enumerated type wrapper generated by the Java Type Library Wizard. If so, you could remove "msado15." from the syntax. However, your class would need to inherit from each Java object and enumerated type interface that it references to completely eliminate the need to reference msado15.* in front of all ADO objects and enumerated values.

For more sample code, see [ADO Java Class Wrappers](#).

To run the Java Type Library Wizard from Visual J++ version 1.x

1. From the **Tools** menu, select **Java Type Library Wizard**.
2. Select "Microsoft ActiveX Data Objects Library" and click **OK**. This now (re)generates files in the \trustlib directory for ADO (by default at c:\winnt\java\trustlib\msado15). If you used the Microsoft SDK for Java to already generate classes for ADO, they will be replaced with those from the Java Type Library Wizard.
3. To use these files, open your project in Visual J++. From the **Project** menu, choose **Add To Project**. Select **Files**, and add all of the .JAVA files generated in the \trustlib directory (by default at c:\winnt\java\trustlib\msado15) to your project.

See Also

[ADO for Windows Foundation Classes](#) | [ADO Java Class Wrappers](#) | [Using ADO with the Microsoft SDK for Java](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Using the Microsoft SDK for Java

The Microsoft SDK for Java is the developer kit for the Microsoft Internet Explorer environment. Tools, information, and samples are provided to help you develop Java programs and applets based on JDK 1.1 and the Microsoft Win32 virtual machine (Microsoft VM). The Microsoft SDK for Java is not tied to Microsoft Visual J++ and can be downloaded from <http://www.microsoft.com/Java>.

The Jactivex.exe utility generates classes from a type library but can only be invoked on the command line. This feature is not integrated with the Visual J++ development environment. Unlike the classes generated by the [Java Type Library Wizard](#), you can step into the class wrappers created by the SDK. This is useful for debugging how your code uses the ADO wrapper classes.

This mechanism reads the ADO type library and generates classes that you can instantiate within your application. It generates those classes in the following location: \<windows directory>\Java\trustlib\msado15.

Creating an ADO application in Java using the Microsoft SDK for Java is fundamentally identical, from the perspective of source code, to using the Java Type Library Wizard. For sample code, see [ADO Java Class Wrappers](#). The only real difference is in how you generate the wrapper classes in the first place, as demonstrated in the steps below.

To create an ADO project with the Microsoft SDK for Java

1. Run the following from a command prompt. You need to set the path to include the Microsoft SDK for Java Bin directory, or run the command from that location. Typically, the Microsoft SDK for Java is installed in the same location as Visual Studio. This is a single command statement.

```
\<path to DevStudio>\<path to Java SDK>\bin\JactiveX.exe /javat1
```

2. Run the following command to compile the generated classes. The /g:t switch turns on generation of debug symbols so that you can trace into the .Java symbols. Remove it for release builds.

```
jvc /g:t c:\<windows>\Java\trustlib\msado15\*.Java
```

3. To use these files, open your project in Visual J++. From the **Project** menu, choose **Add To Project**. Select **Files**, and add all of the .JAVA files generated in the trustlib\msado15 directory to your project.

See Also

[ADO for Windows Foundation Classes](#) | [ADO Java Class Wrappers](#) | [Using ADO with the Java Type Library Wizard](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

ADO Java Class Wrappers

This code declares an instance of the ADO [Recordset](#) class wrapper and initializes it, all on the same line of code. Further, it declares variables for each of the arguments in the [Open](#) method, especially for [LockType](#) and [CursorType](#) (because Java doesn't support enumerated types). It opens and closes the **Recordset** object. Setting Rs1 to NULL merely schedules that variable to be released when Java performs its systematic and intermittent release of unused objects.

```
public static void main( String args[] )
{
    msado15._Recordset  Rs1 = new msado15.Recordset();
    Variant Source      = new Variant( "SELECT * FROM Authors" );
    Variant Connect     = new Variant( "DSN=AdoDemo;UID=admin;PWD=" );
    int    LockType     = msado15.CursorTypeEnum.adOpenForwardOnly;
    int    CursorType   = msado15.LockTypeEnum.adLockReadOnly;
    int    Options      = -1;

    Rs1.Open( Source, Connect, LockType, CursorType, Options );
    Rs1.Close();
    Rs1 = null;

    System.out.println( "Success!\n" );
}
```

See Also

[Using ADO with the Java Type Library Wizard](#) | [Using ADO with the Microsoft SDK for Java](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Appendixes

Appendix D: ADO Samples

The MDAC SDK contains several sample applications that demonstrate usage of ADO and RDS code.

The ADO documentation also contains code examples in multiple languages, which are included as topics within this online help. For more information about these examples, see [ADO Code Examples](#).

Many other sample code resources are available. Visual Studio documentation and the MSDN Library contain a variety of sample applications and documentation that can help you learn about ADO and RDS.

Visual Studio Samples

- [Visual Studio Solutions Center Web site](#)
- [Island Hopper News Sample](#)
- [Visual Basic Samples](#)
- [Visual C++ Samples](#)
- [Visual InterDev Samples](#)
- [Visual J++ Samples](#)

Visual Studio Documentation

- Visual Studio: [Data Access and Security Strategies](#)
- Visual Basic: [Data Access Guide](#)
- Visual InterDev: [Integrating Databases](#)
- Visual J++: [Accessing Data](#)

See Also

[ADO API Reference](#) | [Configuring RDS](#) | [ActiveX Data Objects Start Page](#) | [Programming with ADO](#) | [Appendix A: Providers](#) | [What's New in ADO](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Glossary

ADO Glossary

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

A

[absolute URL](#)

[ActiveX control](#)

[ADISAPI](#)

[Advanced Data Internet Server Application Programming Interface](#)

[aggregate function](#)

[alias](#)

[apartment threading](#)

[asynchronous operation](#)

B

[binding entry](#)

[bitmask](#)

[bookmark](#)

[business object](#)

[business rule](#)

C

[calculated expression](#)

[chapter](#)

[chapter-alias](#)

[character set](#)

[child](#)

[child-alias](#)

[class identifier](#)

[CLSID](#)

[client tier](#)

[COM](#)

[COM component](#)

[comparison operator](#)

[component](#)

[Component Object Model](#)

[compound file](#)

[constant](#)

[cursor](#)

D

[data binding](#)

[data definition language](#)

[data manipulation language](#)

[data provider](#)

[data shaping](#)

[data source name](#)

[data source tier](#)

[DCOM](#)

[DDL](#)

[default stream](#)

[disconnected recordset](#)

[distributed application](#)

[Distributed Component Object Model](#)

[DLL](#)

[DML](#)

[document source provider](#)

[DSN](#)

[dynamic-link library](#)

[dynamic property](#)

E

[enumeration](#)

[event](#)

[event handler](#)

F

G

H

[handler](#)

[hierarchical recordset](#)

[hierarchy](#)

I

[Internet Server Application Programming Interface](#)

[ISAPI](#)

J

K

[key](#)

L

M

[marshaling](#)

[middle tier](#)

[MIME](#)

[Multi-purpose Internet Mail Extension](#)

N

[node](#)

O

[object variable](#)

[ODBC](#)

[OLE DB](#)

[Open Database Connectivity](#)

[optimistic locking](#)

[ordinal value](#)

P

[parameterized command](#)

[parent](#)

[parent-alias](#)

[parent-child relationship](#)

[persist](#)

[pessimistic locking](#)

[pooling](#)

[ProgID](#)

[programmatic identifier](#)

[proxy](#)

Q

R

[relative URL](#)

[remote data source](#)

[resource record](#)

[root](#)

[rowset](#)

S

[schema](#)

[scope](#)

[service provider](#)

[shaped Recordset](#)

[sibling](#)

[stored procedure](#)

[stub](#)

[sub-node](#)

[synchronous operation](#)

T

[tree](#)

U

[Uniform Resource Locator](#)

[URL](#)

V

W

[Web server](#)

X

Y

Z

Terms and Definitions

absolute URL

A fully qualified URL that specifies the location of a resource that resides on the Internet or an intranet. See also [URL](#) and [relative URL](#).

[Return to top](#)

ActiveX control

Self-registering, in-process COM component that often has a visual element either at design time or run time. ActiveX controls also have the ability to communicate with an Active Document container, such as Microsoft Internet Explorer.

[Return to top](#)

ADISAPI (Advanced Data Internet Server Application Programming Interface)

An ISAPI DLL that provides parsing, Automation control, **Recordset** marshaling, and MIME packaging. The ADISAPI component works through the API provided by Internet Information Services (IIS). See also [ISAPI](#).

[Return to top](#)

aggregate function

In a query, a function such as COUNT, AVG, or STDEV that calculates a

value using all the rows in a column of a table. In writing expressions and in programming, you can use SQL aggregate functions (including the three listed above) and domain aggregate functions to determine various statistics.

[Return to top](#)

alias

An alternate name you give to a column or expression in an SQL SELECT statement, often shorter or more meaningful. For example, BobSales is the alias in the following SELECT statement: "Select wr-Sales as BobSales from SalesDB". An alias can be used to dynamically assign columns to control bindings on the **DataControl** object.

[Return to top](#)

apartment threading

A COM threading model where all calls to an object occur on one thread. In apartment threading, COM synchronizes and marshals calls. See also [COM](#).

[Return to top](#)

asynchronous operation

An operation that returns control to the calling program without waiting for the operation to complete. Before the operation is complete, code execution continues. See also [synchronous operation](#).

[Return to top](#)

binding entry

A mapping between a field in a table and a variable. In the ADO Visual C++ extensions, **Recordset** fields are mapped to C/C++ variables.

[Return to top](#)

bitmask

A numeric value intended for a bit-by-bit value comparison with other numeric values, typically to flag options in parameter or return values. Usually this comparison is done with bitwise logical operators, such as **And** and **Or** in Visual Basic, **&** and **|** in C++.

For example, the ADO **FieldAttributeEnum** values can be used as bitmasks to determine the attributes of a field. Suppose you wanted to determine if a field was updateable. You could test for this with the following expression in Visual Basic:

```
Field.Attributes AND adFldUpdatable
```

If the result is TRUE, then the field is updateable.

[Return to top](#)

bookmark

A marker that uniquely identifies a row within a set of rows so that a user can quickly navigate to it.

[Return to top](#)

business object

An object that performs a defined set of operations, such as data validation or business rule logic. Business objects usually reside on the middle tier.

[Return to top](#)

business rule

The combination of validation edits, logon verifications, database lookups, policies, and algorithmic transformations that constitute an enterprise's way of doing business. Also known as *business logic*.

[Return to top](#)

calculated expression

An expression that is not constant, but whose value depends upon other

values. To be evaluated, a calculated expression must obtain and compute values from other sources, typically in other fields or rows.

[Return to top](#)

chapter

A reference to a range of rows from a data source. In ADO, a chapter is typically a reference to another **Recordset**.

Chapter columns make it possible to define a *parent-child* relationship where the *parent* is the **Recordset** containing the chapter column and the *child* is the **Recordset** represented by the chapter.

[Return to top](#)

chapter-alias

An alias that refers to the column appended to the parent.

[Return to top](#)

character set

A mapping of a set of characters to their numeric values. For example, Unicode is a 16-bit character set capable of encoding all known characters and used as a worldwide character-encoding standard.

[Return to top](#)

child

The dependant side of a hierarchical relationship. A child is a node in a hierarchical structure that has another node above it (closer to the root). See also [child-alias](#), [parent-child relationship](#), [parent](#).

[Return to top](#)

child-alias

An alias that refers to the child. See also [alias](#), [child](#).

[Return to top](#)

CLSID (class identifier)

A universally unique identifier (UUID) that identifies a COM component. Each COM component has its CLSID in the Windows Registry so that it can be loaded by other applications. See also [ProgID](#), [COM](#).

[Return to top](#)

client tier

A logical layer of a distributed system that typically presents data to and processes input from the user, sometimes referred to as the *front end*. Usually, the client tier requests data from a server based on input, and then formats and displays the result. See also [middle tier](#), [data source tier](#), [distributed application](#).

[Return to top](#)

COM (Component Object Model)

A binary standard that enables objects to interoperate in a networked environment regardless of the language in which they were developed or on which computers they reside. COM-based technologies include ActiveX Controls, Automation, and object linking and embedding (OLE). COM allows an object to expose its functionality to other components and to host applications. It defines both how the object exposes itself and how this exposure works across processes and across networks. COM also defines the object's life cycle.

[Return to top](#)

COM component

Binary file — such as .dll, .ocx, and some .exe files — that supports the COM standard for providing objects. Such a file contains code for one or more class factories, COM classes, registry-entry mechanisms, loading code, and so on.

[Return to top](#)

comparison operator

An operator that compares two expressions and returns a Boolean value.

A criteria parameter that may be expressed as ">" (greater than), "<" (less than), "=" (equal), ">=" (greater than or equal), "<=" (less than or equal), "<>" (not equal), or "like" (pattern matching).

[Return to top](#)

component

An object that encapsulates both data and code, and provides a well-specified set of publicly available services.

[Return to top](#)

compound file

An implementation of COM structured storage for files. A compound file stores separate objects in a single, structured file consisting of two main elements: storage objects and stream objects. Together, they function like a file system within a file. For more information, see Compound Files in the Microsoft Platform SDK.

A number of individual files bound together in one physical file. Each individual file in a compound file can be accessed as if it were a single physical file.

[Return to top](#)

constant

A numeric or string value that does not change. Named ADO enumerations (enumerated constants) can be used in your code instead of actual values, for example, **adUseClient** is a constant whose value is 3. (Const adUseClient = 3). See also [enumeration](#).

[Return to top](#)

cursor

A database element that controls record navigation, updateability of data, and the visibility of changes made to the database by other users.

[Return to top](#)

data binding

The process of associating the objects or controls of an application to a data source. A control associated with a data source is called a *data-bound control*.

The contents of a data-bound control are associated with values from a database. For example, a grid control that is bound to a **Recordset** object can be updated when the rows in the **Recordset** are updated. When new values are retrieved by the **Recordset**, new values are displayed in the grid.

[Return to top](#)

data provider

Software that exposes data to an ADO application either directly or via a service provider. See also [service provider](#).

[Return to top](#)

data shaping

A technique which makes use of a formalized syntax (called **Shape language**) to define a specialized **Recordset** object (called a [shaped Recordset](#)) that contains not just data, but also references to other **Recordset** objects and/or computed values based on those other **Recordset** objects.

[Return to top](#)

data source tier

A logical layer of a distributed system that represents a computer running a DBMS, such as an SQL Server database. See also [client tier](#), [middle tier](#), [distributed application](#).

[Return to top](#)

DCOM

A wire protocol that enables COM components to communicate directly with each other across a network. See also [COM](#), [component](#).

[Return to top](#)

DDL (Data Definition Language)

Those statements in SQL that define, as opposed to manipulate, data. The schema of a database is created or modified with DDL. For example, **CREATE TABLE**, **CREATE INDEX**, **GRANT**, and **REVOKE** are SQL DDL statements.

[Return to top](#)

default stream

A text or binary stream (represented by a **Stream** object) that is associated with **Record** or **Recordset** objects when using certain OLE DB providers, such as the Microsoft OLE DB Provider for Internet Publishing. The default stream typically contains the contents of a file such as the HTML code for the root of a Web site.

[Return to top](#)

distributed application

A program written so that the processing can be divided across multiple computers over a network. Typically, a distributed application is divided into presentation, business logic, and data store layers, or *tiers*. See also [client tier](#), [middle tier](#), [data source tier](#).

[Return to top](#)

disconnected Recordset

A **Recordset** object in a client cache that no longer has a live connection to the server. If the original data source needs to be accessed again for some reason, such as updating data, the connection must be re-established. However, the collections, properties, and methods of a disconnected **Recordset** can still be accessed.

[Return to top](#)

DLL (dynamic-link library)

A file that contains one or more functions that are compiled, linked, and stored separately from the processes that use them. The operating system maps the DLLs into the address space of the calling process when the process is starting, or while it is running.

[Return to top](#)

DML (Data Manipulation Language)

Those statements in SQL that manipulate, as opposed to define, data. The values in a database are selected and modified with DML. For example, **INSERT**, **UPDATE**, **DELETE**, and **SELECT** are SQL DML statements.

[Return to top](#)

document source provider

A special class of providers that manage folders and documents. When a document is represented by a **Record** object, or a folder of documents is represented by a **Recordset** object, the document source provider populates those objects with a unique set of fields that describe characteristics of the document, instead of the actual document itself. See also [resource record](#).

[Return to top](#)

DSN (data source name)

The collection of information used to connect your application to a

particular ODBC database. The ODBC Driver Manager uses this information to create a connection to the database. A DSN can be stored in a file (a file DSN) or in the Windows Registry (a machine DSN).

[Return to top](#)

dynamic property

A property specific to a data provider or the cursor service. The **Properties** collection of an object is populated with these automatically ("dynamically"). An object has no dynamic properties until it is connected to a data source through a particular data provider. See also [data provider](#), [cursor](#).

[Return to top](#)

enumeration

A list of named constants. Enumerated values need not be unique. However the name of each value must be unique within the scope where the enumeration is defined. In ADO, enumerations are used for numeric parameter and return values, to add meaning to ADO code and to shield the developer from the numeric values (which may change from version to version). For example, to open a static **Recordset**, use the **adOpenStatic** enumerated value:

```
Recordset.Open , , adOpenStatic
```

Also referred to as *enumerated constant*. See also [constant](#).

[Return to top](#)

event

An action recognized by an object, for which you can write code to respond. Events can be generated by command execution, transaction completion, recordset navigation, and data updates, among other actions. See also [event handler](#).

[Return to top](#)

event handler

An event handler is the code that is executed when an event occurs. See also [event](#).

[Return to top](#)

handler

A routine that manages a common and relatively simple condition or operation, such as error recovery or data management.

[Return to top](#)

hierarchical Recordset

A **Recordset** that contains another **Recordset**. See also [data shaping, chapter](#).

For more information, see [Accessing Rows in a Hierarchical Recordset](#)

[Return to top](#)

hierarchy

In general, a hierarchy is a ranked structure with a top level and subordinate levels. In ADO, hierarchical **Recordsets** are used to represent the parent-child relationship between a record and a chapter. Also in ADO, **Record** and **Stream** objects can be used to access hierarchical tree structures such as a folder and documents. ADO MD also includes **Hierarchy** objects to represent a relationship between the levels of a dimension in an OLAP cube. See also [hierarchical Recordsets, parent-child relationship, chapter, tree](#).

[Return to top](#)

ISAPI (Internet Server Application Programming Interface)

A set of functions for Internet servers, such as a Windows NT Server/Windows 2000 Server running Microsoft Internet Information

Services (IIS).

[Return to top](#)

key

A column or columns in a table that uniquely identify a row; often used to index a table.

[Return to top](#)

marshaling

The process of packaging, sending, and unpackaging interface method parameters across thread or process boundaries.

[Return to top](#)

middle tier

The logical layer in a distributed system between a user interface or Web client and the database. This is typically where business objects are instantiated. The middle tier is a collection of business rules and functions that generate and operate upon receiving information. They accomplish this through business rules, which can change frequently, and are thus encapsulated into components that are physically separate from the application logic itself. Also known as *application server tier*. See also [distributed application](#), [client tier](#), [data source tier](#).

[Return to top](#)

MIME (Multi-purpose Internet Mail Extension)

An Internet protocol originally developed to allow exchange of electronic mail messages with rich content across heterogeneous network, machine, and e-mail environments. In practice, MIME has also been adopted and extended by non-mail applications.

MIME is a standard that allows binary data to be published and read on the Internet. The header of a file with binary data contains the MIME type of

the data; this informs client programs (Web browsers and mail packages, for instance) that they will need to handle the data in a different way than they handle straight text. For example, the header of a Web document containing a JPEG graphic contains the MIME type specific to the JPEG file format. This allows a browser to display the file with its JPEG viewer, if one is present.

[Return to top](#)

node

An element in a hierarchical tree structure. A node may be the root, or the child of another node. A node can also be the parent of multiple children. See also [hierarchy](#), [tree](#), [root](#), [child](#), [parent](#).

[Return to top](#)

object variable

A variable that contains a reference to an object. For example, `objCustomObject` is a variable that points to an object of type `CustomObject`:

```
Set objCustomObject = CreateObject(adodb.Recordset)
```

[Return to top](#)

ODBC (Open Database Connectivity)

A standard programming language interface used to connect to a variety of data sources. This is usually accessed through Control Panel, where data source names (DSNs) can be assigned to use specific ODBC drivers.

[Return to top](#)

OLE DB

A set of interfaces that expose data from a variety of sources using COM. OLE DB interfaces provide applications with uniform access to data stored in diverse information sources. These interfaces support the amount of

DBMS functionality appropriate to the data source, enabling it to share its data. See also [COM](#).

[Return to top](#)

optimistic locking

A type of locking in which the data page containing one or more records, including the record being edited, is unavailable to other users only while the record is being updated by the **Update** method, but is available before and after the call to **Update**.

Optimistic locking is used when the **Recordset** object is opened with the **LockType** parameter or property set to **adLockOptimistic** or **adLockBatchOptimistic**. See also [pessimistic locking](#).

[Return to top](#)

ordinal value

The numeric location of an item within an order. In an ADO collection, the ordinal value of the first item is zero (0). The next item is one (1), and so on.

[Return to top](#)

parameterized command

A query or command that allows you to set parameter values before the command is executed. For example, a SQL string can be parameterized by embedding parameter markers in the SQL string (designated by the '?' character). The application then specifies values for each parameter and executes the command.

[Return to top](#)

parent

The controlling side of a hierarchical relationship. In a hierarchical structure, a parent has one or more child nodes directly beneath it in the

hierarchy. See also [parent-alias](#), [parent-child relationship](#), [child](#).

[Return to top](#)

parent-alias

An alias that refers to the parent. See also [alias](#), [parent](#).

[Return to top](#)

parent-child relationship

A relationship in a hierarchical structure in which the parent is one level higher and directly associated with one or more children. A child is one level lower and must have one parent. See also [parent](#), [child](#).

[Return to top](#)

persist

To save data in a permanent state, such as saving a **Recordset** to a file.

[Return to top](#)

pessimistic locking

A type of locking in which the page containing one or more records, including the record being edited, is unavailable to other users to ensure that an update will be made. Pessimistic locking behavior is defined by the OLE DB provider. Typically, records are locked upon editing and remain unavailable until the **Update** method has completed.

Pessimistic locking is enabled when the **Recordset** object is opened with the **LockType** parameter or property set to **adLockPessimistic**. See also [optimistic locking](#).

[Return to top](#)

pooling

A performance optimization based on using collections of pre-allocated

resources, such as objects or database connections. It is more efficient to draw an existing resource from the pool than to create a new resource.

[Return to top](#)

ProgID (programmatic identifier)

A unique name mapped to the Windows registry by a COM application. The ProgID for an ADO Connection is "ADODB.Connection". See also [CLSID](#), [COM](#).

[Return to top](#)

proxy

An interface-specific object that provides the parameter marshaling and communication required for a client to call an application object that is running in a different execution environment, such as on a different thread or in another process. The proxy is located with the client and communicates with a corresponding stub that is located with the application object that is being called. See also [stub](#).

[Return to top](#)

relative URL

A partially qualified URL that specifies a resource on the Internet or an intranet whose location is relative to a starting point specified by an absolute URL or equivalent ADO Connection object. In effect, the concatenated absolute and relative URLs constitute a complete URL. See also [URL](#) and [absolute URL](#).

[Return to top](#)

remote data source

A data source that exists on a another computer, rather than on the local system (where the client application runs).

[Return to top](#)

resource record

A record from a document source provider that contains fields for the definition and description of a folder or document. The document itself is not contained in the resource record but typically can be accessed by the default stream or a field in the resource record containing a URL. See also [document source provider](#), [default stream](#), [URL](#).

[Return to top](#)

root

The top level in a hierarchical tree structure. The root node has no parents, but may have children. See also [hierarchy](#), [tree](#), [parent](#), [child](#).

[Return to top](#)

rowset

A set of rows from a data source, all having the same field schema. A rowset can represent all or some fields from a table. A rowset can also represent a virtual table, created by a query or a join of two or more tables. In ADO, rowsets are represented by **Recordset** objects.

[Return to top](#)

schema

A description of a database to the database management system (DBMS), typically generated using the data definition language provided by the DBMS. A schema defines attributes of the database, such as tables, columns, and properties.

[Return to top](#)

scope

The range of reference for an object or variable or a range of records in a view or table. For example, local variables can be referenced only within the procedure in which they were defined. Public variables are accessible

from anywhere in the application. Objects, such as the current database, are in scope if they are in the defined search path. Record ranges can be specified with a Scope clause in many commands.

[Return to top](#)

service provider

Software that encapsulates a service by producing and consuming data, augmenting features in your ADO applications. It is a provider that does not directly expose data, rather it provides a service, such as query processing. The service provider may process data provided by a data provider. See also [data provider](#).

[Return to top](#)

shaped Recordset

A **Recordset** whose columns have been specifically defined to contain not just data, but also references (called [chapters](#)) to other **Recordset** objects and/or computed values based on other **Recordset** objects.

[Return to top](#)

sibling

Any two or more nodes in a hierarchical structure that are on the same level in the hierarchy. The root node in a hierarchy has no siblings.

[Return to top](#)

stored procedure

A precompiled collection of code such as SQL statements and optional control-of-flow statements stored under a name and processed as a unit. Stored procedures are stored within a database; they can be executed with one call from an application and allow user-declared variables, conditional execution, and other powerful programming features.

[Return to top](#)

stub

An interface-specific object that provides the parameter marshaling and communication required for an application object to receive calls from a client that is running in a different execution environment, such as on a different thread or in another process. The stub is located with the application object and communicates with a corresponding proxy that is located with the client that calls it. See also [proxy](#).

[Return to top](#)

sub-node

See [child](#).

[Return to top](#)

synchronous operation

An operation initiated by code that completes before the next operation may start. See also [asynchronous operation](#).

[Return to top](#)

tree

A structure representing a hierarchical relationship between elements (nodes). There is one node at the top level of a tree (the root). Underneath the root, there can be multiple children. Each child may in turn be the parent of other children, thus branching like a tree. A folder containing documents and other folders is a typical example of a tree structure. See also [hierarchy](#), [node](#), [root](#), [child](#), [parent](#).

[Return to top](#)

URL (Uniform Resource Locator)

Specifies the location of a resource residing on the Internet or an intranet. A complete URL consists of a scheme (such as FTP, HTTP, mailto, file, and so on), followed by a colon, a server name, and the full path of a resource

(such as a document, graphic, or other file). Some examples of URLs are:

```
http://www.domain.com/default.html  
ftp://ftp.server.somewhere/ftp.file  
file://Server/Share/File.doc
```

See also [absolute URL](#) and [relative URL](#).

[Return to top](#)

Web server

A computer that provides Web services and pages to intranet and Internet users.

[Return to top](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Programmer's Reference

ADO Programmer's Reference

This section of the ADO documentation contains reference topics for ADO, RDS, ADO MD, and ADOX.

- [ADO API Reference](#)
- [RDS API Reference](#)
- [ADO MD API Reference](#)
- [ADOX API Reference](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO API Reference

This section of the ADO documentation contains topics for each ADO object, collection, property, dynamic property, method, event, and enumeration. In addition, it contains a list of ADO syntax indexes to be used with Microsoft Visual C++ and Windows Foundation Classes (WFC).

For more information, search for a specific topic in the index or refer to the following topics:

- [ADO Object Model](#)
- [ADO Objects](#)
- [ADO Collections](#)
- [ADO Properties](#)
- [ADO Dynamic Properties](#)
- [ADO Methods](#)
- [ADO Events](#)
- [ADO Enumerated Constants](#)
- [ADO Syntax Indexes](#)

See Also

[Appendix D: ADO Samples](#) | [Configuring RDS](#) | [ActiveX Data Objects Start Page](#) | [Appendix C: Programming with ADO](#) | [Appendix A: Providers](#) | [What's New in ADO](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Object Model

The following figures show the ADO objects and their collections. Click an object or collection for more information.

Connection



See Also

[ADO Collections](#) | [ADO Dynamic Properties](#) | [ADO Enumerated Constants](#) | [Appendix B: ADO Errors](#) | [ADO Events](#) | [ADO Methods](#) | [ADO Objects](#) | [ADO Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Objects and Interfaces

The relationships between these objects are represented in the [ADO Object Model](#).

Each object can be contained in its corresponding collection. For example, an [Error](#) object can be contained in an [Errors](#) collection. For more information, see [ADO Collections](#) or a specific collection topic.

ADORecordConstruction	Constructs an ADO Record object from an OLE DB Row object in a C/C++ application.
ADORecordsetConstruction	Constructs an ADO Recordset object from an OLE DB Rowset object in a C/C++ application.
Error	Contains details about data access errors that pertain to a single operation involving the provider.
Field	Represents a column of data with a common data type.
Parameter	Represents a parameter or argument associated with a Command object based on a parameterized query or stored procedure.
Property	Represents a dynamic characteristic of an ADO object that is defined by the provider.
Record	Represents a row of a Recordset , or a directory or file in a file system.
Recordset	Represents the entire set of records from a base table or the results of an executed command. At any time, the Recordset object refers to only a single record within the set as the current record.
Stream	Represents a binary stream of data.

See Also

[ADO API Reference](#) | [ADO Collections](#) | [ADO Dynamic Properties](#) | [ADO Enumerated Constants](#) | [Appendix B: ADO Errors](#) | [ADO Events](#) | [ADO Methods](#) | [ADO Object Model](#) | [ADO Properties](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

ADORecordConstruction Interface

The **ADORecordConstruction** interface is used to construct an ADO **Record** object from an OLE DB **Row** object in a C/C++ application.

This interface supports the following properties:

Properties

[ParentRow](#)

Write-only.

Sets the container of an OLE DB **Row** object on this ADO **Record** object.

[Row](#)

Read/Write.

Gets/sets an OLE DB **Row** object from/on this ADO **Record** object.

Methods

None.

Events

None.

Remarks

Given an OLE DB **Row** object (pRow), the construction of an ADO **Record** object (adoR), amounts to the following three basic operations:

1. Create an ADO **Record** object:

```
_RecordPtr adoR;  
adoRs.CreateInstance(__uuidof(_Record));
```

2. Query the **IADORecordConstruction** interface on the **Record** object:

```
adoRecordConstructionPtr adoRConstruct=NULL;
adoR->QueryInterface(__uuidof(ADORecordConstruction),
                    (void**)&adoRConstruct);
```

3. Call the **IADORecordConstruction::put_Row** property method to set the OLE DB **Row** object on the ADO **Record** object:

```
IUnknown *pUnk=NULL;
pRow->QueryInterface(IID_IUnknown, (void**)&pUnk);
adoRConstruct->put_Row(pUnk);
```

The resultant **adoR** object now represents the ADO **Record** object constructed from the OLE DB **Row** object.

An ADO **Record** object can also be constructed from the container of an OLE DB **Row** object.

Requirements

Version: ADO 2.0 and later

Library: msado15.dll

UUID: 00000567-0000-0010-8000-00AA006D2EA4

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADORecordsetConstruction Interface

The **ADORecordsetConstruction** interface is used to construct an ADO **Recordset** object from an OLE DB **Rowset** object in a C/C++ application.

This interface supports the following properties:

Properties

[Chapter](#)

Read/Write.

Gets/sets an OLE DB **Chapter** object from/on this ADO **Recordset** object.

[RowPosition](#)

Read/Write.

Gets/sets an OLE DB **RowPosition** object from/on this ADO **Recordset** object.

[Rowset](#)

Read/Write.

Gets/sets an OLE DB **Rowset** object from/on this ADO **Recordset** object.

Methods

None.

Events

None.

Remarks

Given an OLE DB **Rowset** object (pRowset), the construction of an ADO **Recordset** object (adoRs) amounts to the following three basic operations:

1. Create an ADO **Recordset** object:

```
Recordset20Ptr adoRs;  
adoRs.CreateInstance(__uuidof(Recordset));
```

2. Query the **IADORecordsetConstruction** interface on the **Recordset** object:

```
adoRecordsetConstructionPtr adoRsConstruct=NULL;  
adoRs->QueryInterface(__uuidof(ADORecordsetConstruction),  
                      (void**)&adoRsConstruct);
```

3. Call the **IADORecordsetConstruction::put_Rowset** property method to set the OLE DB Rowset object on the ADO Recordset object:

```
IUnknown *pUnk=NULL;  
pRowset->QueryInterface(IID_IUnknown, (void**)&pUnk);  
adoRsConstruct->put_Rowset(pUnk);
```

The resultant `adoRs` object now represents the ADO **Recordset** object constructed from the OLE DB **Rowset** object.

You can also construct an ADO **Recordset** object from an OLE DB **Chapter** or **RowPosition** object.

Requirements

Version: ADO 2.0 and later

Library: msado15.dll

UUID: 00000283-0000-0010-8000-00AA006D2EA4

See Also

[Recordset Object](#) | [Rowset Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Command Object

Defines a specific command that you intend to execute against a data source.

Command



Remarks

Use a **Command** object to query a database and return records in a [Recordset](#) object, to execute a bulk operation, or to manipulate the structure of a database. Depending on the functionality of the provider, some **Command** collections, methods, or properties may generate an error when referenced.

With the collections, methods, and properties of a **Command** object, you can do the following:

- Define the executable text of the command (for example, an SQL statement) with the [CommandText](#) property.
- Define [parameterized](#) queries or stored-procedure arguments with [Parameter](#) objects and the [Parameters](#) collection.
- Execute a command and return a **Recordset** object if appropriate with the [Execute](#) method.
- Specify the type of command with the [CommandType](#) property prior to execution to optimize performance.
- Control whether the provider saves a prepared (or compiled) version of the command prior to execution with the [Prepared](#) property.
- Set the number of seconds that a provider will wait for a command to execute with the [CommandTimeout](#) property.
- Associate an open connection with a **Command** object by setting its [ActiveConnection](#) property.
- Set the [Name](#) property to identify the **Command** object as a method on the associated [Connection](#) object.
- Pass a **Command** object to the [Source](#) property of a **Recordset** in order to obtain data.

- Access provider-specific attributes with the [Properties](#) collection.

Note To execute a query without using a **Command** object, pass a query string to the [Execute](#) method of a **Connection** object or to the [Open](#) method of a **Recordset** object. However, a **Command** object is required when you want to [persist](#) the command text and re-execute it, or use query parameters.

To create a **Command** object independently of a previously defined **Connection** object, set its **ActiveConnection** property to a valid connection string. ADO still creates a **Connection** object, but it doesn't assign that object to an [object variable](#). However, if you are associating multiple **Command** objects with the same connection, you should explicitly create and open a **Connection** object; this assigns the **Connection** object to an object variable. If you do not set the **Command** object's **ActiveConnection** property to this object variable, ADO creates a new **Connection** object for each **Command** object, even if you use the same connection string.

To execute a **Command**, simply call it by its [Name](#) property on the associated **Connection** object. The **Command** must have its **ActiveConnection** property set to the **Connection** object. If the **Command** has parameters, pass their values as arguments to the method.

If two or more **Command** objects are executed on the same connection and either **Command** object is a stored procedure with output parameters, an error occurs. To execute each **Command** object, use separate connections or disconnect all other **Command** objects from the connection.

The **Parameters** collection is the default member of the **Command** object. As a result, the following two code statements are equivalent.

```
objCmd.Parameters.Item(0)  
objCmd(0)
```

See Also

[Command Object Properties, Methods, and Events](#) | [Connection Object](#) | [Parameters Collection](#) | [Properties Collection](#) | [Appendix A: Providers](#)

ADO 2.5 API Reference

Command Object Properties, Methods, and Events

Properties/Collections

[ActiveConnection Property](#)

[CommandText Property](#)

[CommandTimeout Property](#)

[CommandType Property](#)

[Name Property](#)

[Parameters Collection](#)

[Prepared Property](#)

[Properties Collection](#)

[State Property](#)

Methods

[Cancel Method](#)

[CreateParameter Method](#)

[Execute Method \(ADO Command\)](#)

Events

None.

See Also

[Command Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Connection Object

Represents an open connection to a data source.

Connection



Remarks

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. Depending on the functionality supported by the provider, some collections, methods, or properties of a **Connection** object may not be available.

With the collections, methods, and properties of a **Connection** object, you can do the following:

- Configure the connection before opening it with the [ConnectionString](#), [ConnectionTimeout](#), and [Mode](#) properties. **ConnectionString** is the default property of the **Connection** object.
- Set the [CursorLocation](#) property to client to invoke the [Microsoft Cursor Service for OLE DB](#), which supports batch updates.
- Set the default database for the connection with the [DefaultDatabase](#) property.
- Set the level of isolation for the transactions opened on the connection with the [IsolationLevel](#) property.
- Specify an OLE DB provider with the [Provider](#) property.
- Establish, and later break, the physical connection to the data source with the [Open](#) and [Close](#) methods.
- Execute a command on the connection with the [Execute](#) method and configure the execution with the [CommandTimeout](#) property.

Note To execute a query without using a Command object, pass a query string to the **Execute** method of a **Connection** object. However,

a [Command](#) object is required when you want to [persist](#) the command text and re-execute it, or use query parameters.

- Manage transactions on the open connection, including nested transactions if the provider supports them, with the [BeginTrans](#), [CommitTrans](#), and [RollbackTrans](#) methods and the [Attributes](#) property.
- Examine errors returned from the data source with the [Errors](#) collection.
- Read the version from the ADO implementation used with the [Version](#) property.
- Obtain schema information about your database with the [OpenSchema](#) method.

You can create **Connection** objects independently of any other previously defined object.

You can execute commands or stored procedures as if they were native methods on the **Connection** object, as illustrated below.

Execute a command as a native method of a Connection object

To execute a command, give the command a name using the **Command** object [Name](#) property. Set the **Command** object's **ActiveConnection** property to the connection. Then issue a statement where the command name is used as if it were a method on the **Connection** object, followed by any parameters, then followed by a **Recordset** object if any rows are returned. Set the **Recordset** properties to customize the resulting **Recordset**. For example:

```
Dim cnn As New ADODB.Connection
Dim cmd As New ADODB.Command
Dim rst As New ADODB.Recordset
...
cnn.Open "...
cmd.Name = "yourCommandName"
cmd.ActiveConnection = cnn
...
'Your command name, any parameters, and an optional Recordset.
cnn.yourCommandName "parameter", rst
```

Execute a stored procedure as a native method of a Connection object

To execute a stored procedure, issue a statement where the stored procedure name is used as if it were a method on the **Connection** object, followed by any

parameters. ADO will make a "best guess" of parameter types. For example:

```
Dim cnn As New ADODB.Connection
...
'Your stored procedure name and any parameters.
cnn.sp_yourStoredProcedureName "parameter"
```

See Also

[Connection Object Properties, Methods, and Events](#) | [Command Object](#) | [Errors Collection](#) | [Properties Collection](#) | [Recordset Object](#) | [Appendix A: Providers](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Connection Object Properties, Methods, and Events

Properties/Collections

[Attributes Property](#)

[CommandTimeout Property](#)

[ConnectionString Property](#)

[ConnectionTimeout Property](#)

[CursorLocation Property](#)

[DefaultDatabase Property](#)

[Errors Collection](#)

[IsolationLevel Property](#)

[Mode Property](#)

[Properties Collection](#)

[Provider Property](#)

[State Property](#)

[Version Property](#)

Methods

[BeginTrans, CommitTrans, and RollbackTrans Methods](#)

[Cancel Method](#)

[Close Method](#)

[Execute Method \(ADO Connection\)](#)

[Open Method \(ADO Connection\)](#)

[OpenSchema Method](#)

Events

[BeginTransComplete, CommitTransComplete, and RollbackTransComplete Events](#)

[ConnectComplete and Disconnect Events](#)

[ExecuteComplete Event](#)

[InfoMessage Event](#)

[WillConnect Event](#)

[WillExecute Event](#)

See Also

[Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Error Object

Contains details about data access errors that pertain to a single operation involving the provider.

Connection



Remarks

Any operation involving ADO objects can generate one or more provider errors. As each error occurs, one or more **Error** objects are placed in the [Errors](#) collection of the [Connection](#) object. When another ADO operation generates an error, the **Errors** collection is cleared, and the new set of **Error** objects is placed in the **Errors** collection.

Note Each **Error** object represents a specific provider error, not an ADO error. ADO errors are exposed to the run-time exception-handling mechanism. For example, in Microsoft Visual Basic, the occurrence of an ADO-specific error will trigger an **On Error** event and appear in the **Error** object. For a complete list of ADO errors, see the [ErrorValueEnum](#) topic.

You can read an **Error** object's properties to obtain specific details about each error, including the following:

- The [Description](#) property, which contains the text of the error. This is the default property.
- The [Number](#) property, which contains the **Long** integer value of the error constant.
- The [Source](#) property, which identifies the object that raised the error. This is particularly useful when you have several **Error** objects in the **Errors** collection following a request to a data source.
- The [SQLState](#) and [NativeError](#) properties, which provide information from SQL data sources.

When a provider error occurs, it is placed in the **Errors** collection of the

Connection object. ADO supports the return of multiple errors by a single ADO operation to allow for error information specific to the provider. To obtain this rich error information in an error handler, use the appropriate error-trapping features of the language or environment you are working with, then use nested loops to enumerate the properties of each **Error** object in the **Errors** collection.

Microsoft Visual Basic and VBScript Users If there is no valid **Connection** object, you will need to retrieve error information from the **Error** object.

Just as providers do, ADO clears the **OLE Error Info** object before making a call that could potentially generate a new provider error. However, the **Errors** collection on the **Connection** object is cleared and populated only when the provider generates a new error, or when the [Clear](#) method is called.

Some properties and methods return warnings that appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the [Resync](#), [UpdateBatch](#), or [CancelBatch](#) methods on a [Recordset](#) object; the [Open](#) method on a **Connection** object; or set the [Filter](#) property on a **Recordset** object, call the **Clear** method on the **Errors** collection. That way, you can read the [Count](#) property of the **Errors** collection to test for returned warnings.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Error Object Properties, Methods, and Events](#) | [Connection Object](#) | [Errors Collection](#) | [Appendix A: Providers](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Error Object Properties, Methods, and Events

Properties

[Description Property](#)

[HelpContext, HelpFile Properties](#)

[NativeError Property](#)

[Number Property](#)

[Source Property \(ADO Error\)](#)

[SQLState Property](#)

Methods

None.

Events

None.

See Also

[Error Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Field Object

Represents a column of data with a common data type.



Remarks

Each **Field** object corresponds to a column in the [Recordset](#). You use the [Value](#) property of **Field** objects to set or return data for the current record. Depending on the functionality the provider exposes, some collections, methods, or properties of a **Field** object may not be available.

With the collections, methods, and properties of a **Field** object, you can do the following:

- Return the name of a field with the [Name](#) property.
- View or change the data in the field with the **Value** property. **Value** is the default property of the **Field** object.
- Return the basic characteristics of a field with the [Type](#), [Precision](#), and [NumericScale](#) properties.
- Return the declared size of a field with the [DefinedSize](#) property.
- Return the actual size of the data in a given field with the [ActualSize](#) property.
- Determine what types of functionality are supported for a given field with the [Attributes](#) property and [Properties](#) collection.
- Manipulate the values of fields containing long binary or long character data with the [AppendChunk](#) and [GetChunk](#) methods.
- If the provider supports batch updates, resolve discrepancies in field values during batch updating with the [OriginalValue](#) and [UnderlyingValue](#) properties.

All of the metadata properties (**Name**, **Type**, **DefinedSize**, **Precision**, and

NumericScale) are available before opening the **Field** object's **Recordset**. Setting them at that time is useful for dynamically constructing forms.

See Also

[Field Object Properties, Methods, and Events](#) | [Fields Collection](#) | [Properties Collection](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Field Object Properties, Methods, and Events

Properties/Collections

[ActualSize Property](#)

[Attributes Property](#)

[DefinedSize Property](#)

[Name Property](#)

[NumericScale Property](#)

[OriginalValue Property](#)

[Precision Property](#)

[Properties Collection](#)

[Status Property \(ADO Field\)](#)

[Type Property](#)

[UnderlyingValue Property](#)

[Value Property](#)

Methods

[AppendChunk Method](#)

[GetChunk Method](#)

Events

None.

See Also

[Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Parameter Object

Represents a parameter or argument associated with a [Command](#) object based on a parameterized query or stored procedure.

Command



Remarks

Many providers support [parameterized commands](#). These are commands in which the desired action is defined once, but variables (or parameters) are used to alter some details of the command. For example, an SQL SELECT statement could use a parameter to define the matching criteria of a WHERE clause, and another to define the column name for a SORT BY clause.

Parameter objects represent parameters associated with parameterized queries, or the in/out arguments and the return values of stored procedures. Depending on the functionality of the provider, some collections, methods, or properties of a **Parameter** object may not be available.

With the collections, methods, and properties of a **Parameter** object, you can do the following:

- Set or return the name of a parameter with the [Name](#) property.
- Set or return the value of a parameter with the [Value](#) property. **Value** is the default property of the **Parameter** object.
- Set or return parameter characteristics with the [Attributes](#), [Direction](#), [Precision](#), [NumericScale](#), [Size](#), and [Type](#) properties.
- Pass long binary or character data to a parameter with the [AppendChunk](#) method.
- Access provider-specific attributes with the [Properties](#) collection.

If you know the names and properties of the parameters associated with the stored procedure or parameterized query you wish to call, you can use the [CreateParameter](#) method to create **Parameter** objects with the appropriate

property settings and use the [Append](#) method to add them to the [Parameters](#) collection. This lets you set and return parameter values without having to call the [Refresh](#) method on the **Parameters** collection to retrieve the parameter information from the provider, a potentially resource-intensive operation.

See Also

[Parameter Object Properties, Methods, and Events](#) | [Command Object](#) | [CreateParameter Method](#) | [Parameters Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Parameter Object Properties, Methods, and Events

Properties/Collections

[Attributes Property](#)

[Direction Property](#)

[Name Property](#)

[NumericScale Property](#)

[Precision Property](#)

[Properties Collection](#)

[Size Property](#)

[Type Property](#)

[Value Property](#)

Methods

[AppendChunk Method](#)

Events

None.

See Also

[Parameter Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Property Object

Represents a dynamic characteristic of an ADO object that is defined by the provider.

Connection



Remarks

ADO objects have two types of properties: built-in and dynamic.

Built-in properties are those properties implemented in ADO and immediately available to any new object, using the `MyObject.Property` syntax. They do not appear as **Property** objects in an object's [Properties](#) collection, so although you can change their values, you cannot modify their characteristics.

Dynamic properties are defined by the underlying [data provider](#), and appear in the **Properties** collection for the appropriate ADO object. For example, a property specific to the provider may indicate if a [Recordset](#) object supports transactions or updating. These additional properties will appear as **Property** objects in that **Recordset** object's **Properties** collection. Dynamic properties can be referenced only through the collection, using the `MyObject.Properties(0)` or `MyObject.Properties("Name")` syntax.

You cannot delete either kind of property.

A dynamic **Property** object has four built-in properties of its own:

- The [Name](#) property is a string that identifies the property.
- The [Type](#) property is an integer that specifies the property data type.
- The [Value](#) property is a variant that contains the property setting. **Value** is the default property for a **Property** object.

- The [Attributes](#) property is a long value that indicates characteristics of the property specific to the provider.

See Also

[Property Object Properties, Methods, and Events](#) | [Command Object](#) | [Connection Object](#) | [Field Object](#) | [Properties Collection](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Property Object Properties, Methods, and Events

Properties

[Attributes Property](#)

[Name Property](#)

[Type Property](#)

[Value Property](#)

Methods

None.

Events

None.

See Also

[Property Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Record Object

Represents a row from a [Recordset](#) or the data provider, or an object returned by a semi-structured data provider, such as a file or directory.

Record



Remarks

A **Record** object represents one row of data, and has some conceptual similarities with a one-row **Recordset**. Depending upon the capabilities of your provider, **Record** objects may be returned directly from your provider instead of a one-row **Recordset**, for example when an SQL query that selects only one row is executed. Or, a **Record** object can be obtained directly from a **Recordset** object. Or, a **Record** can be returned directly from a provider to semi-structured data, such as the Microsoft Exchange OLE DB provider.

You can view the fields associated with the **Record** object by way of the [Fields](#) collection on the **Record** object. ADO allows object-valued columns including **Recordset**, **SafeArray**, and scalar values in the **Fields** collection of **Record** objects.

If the **Record** object represents a row in a **Recordset**, then it is possible to return to that original **Recordset** with the [Source](#) property.

The **Record** object can also be used by semi-structured data providers such as the [Microsoft OLE DB Provider for Internet Publishing](#), to model tree-structured namespaces. Each node in the tree is a **Record** object with associated columns. The columns can represent the attributes of that node and other relevant information. The **Record** object can represent both a leaf node and a non-leaf node in the tree structure. Non-leaf nodes have other nodes as their contents while leaf nodes do not have such contents. Leaf nodes typically contain binary streams of data while non-leaf nodes may also have a default binary stream associated with them. Properties on the **Record** object identify the type of node.

The **Record** object also represents an alternative way for navigating hierarchically organized data. A **Record** object may be created to represent the root of a specific sub-tree in a large tree structure and new **Record** objects may be opened to represent child nodes.

A resource (for example, a file or directory) can be uniquely identified by an absolute URL. A [Connection](#) object is implicitly created and set to the **Record** object when the **Record** is opened with an absolute URL. A **Connection** object may explicitly be set to the **Record** object via the [ActiveConnection](#) property. The files and directories accessible via the **Connection** object define the *context* in which **Record** operations may occur.

Data modification and navigation methods on the **Record** object also accept a relative URL, which locates a resource using an absolute URL or the **Connection** object context as a starting point.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

A **Connection** object is associated with each **Record** object. Therefore, **Record** object operations can be part of a transaction by invoking **Connection** object transaction methods.

The **Record** object does not support ADO events, and therefore will not respond to notifications.

With the methods and properties of a **Record** object, you can do the following:

- Set or return the associated **Connection** object with the [ActiveConnection](#) property.
- Indicate access permissions with the [Mode](#) property.
- Return the URL of the directory, if any, that contains the resource represented by the **Record** with the [ParentURL](#) property.
- Indicate the absolute URL, relative URL, or **Recordset** from which the **Record** is derived with the [Source](#) property.
- Indicate the current status of the **Record** with the [State](#) property.
- Indicate the type of **Record**—*simple*, *collection*, or *structured document*—with the [RecordType](#) property.
- Halt execution of an asynchronous operation with the [Cancel](#) method.

- Disassociate the **Record** from a data source with the [Close](#) method.
- Copy the file or directory represented by a **Record** to another location with the [CopyRecord](#) method.
- Delete the file, or directory and subdirectories, represented by a **Record** with the [DeleteRecord](#) method.
- Open a **Recordset** containing rows that represent the subdirectories and files of the entity represented by the **Record** with the [GetChildren](#) method.
- Move (rename) the file, or directory and subdirectories, represented by a **Record** to another location with the [MoveRecord](#) method.
- Associate the **Record** with an existing data source, or create a new file or directory with the [Open](#) method.

See Also

[Visual Basic Example](#) | [Visual Basic Example](#)

[Record Object Properties, Methods, and Events](#) | [Fields Collection](#) | [Properties Collection](#) | [Chapter 10: Records and Streams](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Record Object Properties, Methods, and Events

Properties/Collections

[ActiveConnection Property](#)

[Fields Collection](#)

[Mode Property](#)

[ParentURL Property](#)

[Properties Collection](#)

[RecordType Property](#)

[Source Property \(ADO Record\)](#)

[State Property](#)

Methods

[Cancel Method](#)

[Close Method](#)

[CopyRecord Method](#)

[DeleteRecord Method](#)

[GetChildren Method](#)

[MoveRecord Method](#)

[Open Method \(ADO Record\)](#)

Events

None.

See Also

[Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Recordset Object

Represents the entire set of records from a base table or the results of an executed command. At any time, the **Recordset** object refers to only a single record within the set as the current record.

Recordset



Remarks

You use **Recordset** objects to manipulate data from a [provider](#). When you use ADO, you manipulate data almost entirely using **Recordset** objects. All **Recordset** objects consist of records (rows) and fields (columns). Depending on the functionality supported by the provider, some **Recordset** methods or properties may not be available.

ADODB.Recordset is the ProgID that should be used to create a **Recordset** object. Existing applications that reference the outdated ADOR.Recordset ProgID will continue to work without recompiling, but new development should reference ADODB.Recordset.

There are four different [cursor](#) types defined in ADO:

- **Dynamic cursor** — allows you to view additions, changes, and deletions by other users; allows all types of movement through the **Recordset** that doesn't rely on bookmarks; and allows bookmarks if the provider supports them.
- **Keyset cursor** — behaves like a dynamic cursor, except that it prevents you from seeing records that other users add, and prevents access to records that other users delete. Data changes by other users will still be visible. It always supports bookmarks and therefore allows all types of movement through the **Recordset**.
- **Static cursor** — provides a static copy of a set of records for you to use to find data or generate reports; always allows bookmarks and therefore

allows all types of movement through the **Recordset**. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a [client-side Recordset](#) object.

- **Forward-only cursor** — allows you to only scroll forward through the **Recordset**. Additions, changes, or deletions by other users will not be visible. This improves performance in situations where you need to make only a single pass through a **Recordset**.

Set the [CursorType](#) property prior to opening the **Recordset** to choose the cursor type, or pass a *CursorType* argument with the [Open](#) method. Some providers don't support all cursor types. Check the documentation for the provider. If you don't specify a cursor type, ADO opens a forward-only cursor by default.

If the [CursorLocation](#) property is set to **adUseClient** to open a **Recordset**, the **UnderlyingValue** property on [Field](#) objects is not available in the returned **Recordset** object. When used with some [providers](#) (such as the Microsoft ODBC Provider for OLE DB in conjunction with Microsoft SQL Server), you can create **Recordset** objects independently of a previously defined [Connection](#) object by passing a connection string with the **Open** method. ADO still creates a [Connection](#) object, but it doesn't assign that object to an [object variable](#). However, if you are opening multiple **Recordset** objects over the same connection, you should explicitly create and open a **Connection** object; this assigns the **Connection** object to an object variable. If you do not use this object variable when opening your **Recordset** objects, ADO creates a new **Connection** object for each new **Recordset**, even if you pass the same connection string.

You can create as many **Recordset** objects as needed.

When you open a **Recordset**, the current record is positioned to the first record (if any) and the [BOF](#) and [EOF](#) properties are set to **False**. If there are no records, the **BOF** and **EOF** property settings are **True**.

You can use the [MoveFirst](#), **MoveLast**, **MoveNext**, and **MovePrevious** methods; the [Move](#) method; and the [AbsolutePosition](#), [AbsolutePage](#), and [Filter](#) properties to reposition the current record, assuming the provider supports the relevant functionality. Forward-only **Recordset** objects support only the [MoveNext](#) method. When you use the **Move** methods to visit each record (or enumerate the **Recordset**), you can use the **BOF** and **EOF** properties to determine if you've moved beyond the beginning or end of the **Recordset**.

Recordset objects can support two types of updating: immediate and batched. In immediate updating, all changes to data are written immediately to the underlying data source once you call the [Update](#) method. You can also pass arrays of values as parameters with the [AddNew](#) and **Update** methods and simultaneously update several fields in a record.

If a provider supports batch updating, you can have the provider cache changes to more than one record and then transmit them in a single call to the database with the [UpdateBatch](#) method. This applies to changes made with the **AddNew**, **Update**, and [Delete](#) methods. After you call the **UpdateBatch** method, you can use the [Status](#) property to check for any data conflicts in order to resolve them.

Note To execute a query without using a [Command](#) object, pass a query string to the **Open** method of a **Recordset** object. However, a **Command** object is required when you want to [persist](#) the command text and re-execute it, or use query parameters.

The [Mode](#) property governs access permissions.

The **Fields** collection is the default member of the **Recordset** object. As a result, the following two code statements are equivalent.

```
Debug.Print objRs.Fields.Item(0) ' Both statements print  
Debug.Print objRs(0)           ' the Value of Item(0).
```

See Also

[Recordset Object Properties, Methods, and Events](#) | [Connection Object](#) | [Fields Collection](#) | [Properties Collection](#) | [Appendix A: Providers](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Recordset Object Properties, Methods, and Events

Properties/Collections

[AbsolutePage Property](#)

[AbsolutePosition Property](#)

[ActiveCommand Property](#)

[ActiveConnection Property](#)

[BOF, EOF Properties](#)

[Bookmark Property](#)

[CacheSize Property](#)

[CursorLocation Property](#)

[CursorType Property](#)

[DataMember Property](#)

[DataSource Property](#)

[EditMode Property](#)

[Fields Collection](#)

[Filter Property](#)

[Index Property](#)

[LockType Property](#)

[MarshalOptions Property](#)

[MaxRecords Property](#)

[PageCount Property](#)

[PageSize Property](#)

[Properties Collection](#)

[RecordCount Property](#)

[Sort Property](#)

[Source Property \(ADO Recordset\)](#)

[State Property](#)

[Status Property \(ADO Recordset\)](#)

[StayInSync Property](#)

Methods

[AddNew Method](#)

[Cancel Method](#)

[CancelBatch Method](#)

[CancelUpdate Method](#)

[Clone Method](#)

[Close Method](#)

[CompareBookmarks Method](#)

[Delete Method \(ADO Recordset\)](#)

[Find Method](#)

[GetRows Method](#)

[GetString Method](#)

[Move Method](#)

[MoveFirst, MoveLast, MoveNext, and MovePrevious Methods](#)

[NextRecordset Method](#)

[Open Method \(ADO Recordset\)](#)

[Requery Method](#)

[Resync Method](#)

[Save Method](#)

[Seek Method](#)

[Supports Method](#)

[Update Method](#)

[UpdateBatch Method](#)

Events

[EndOfRecordset Event](#)

[FetchComplete Event](#)

[FetchProgress Event](#)

[WillChangeField and FieldChangeComplete Events](#)

[WillChangeRecord and RecordChangeComplete Events](#)

[WillChangeRecordset and RecordsetChangeComplete Events](#)

[WillMove and MoveComplete Events](#)

See Also

[Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Stream Object

Represents a stream of binary data or text.

Stream

Remarks

In tree-structured hierarchies such as a file system or an e-mail system, a [Record](#) 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 in these ways:

- 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.

With the methods and properties of a **Stream** object, you can do the following:

- Open a **Stream** object from a **Record** or URL with the [Open](#) method.
- Close a **Stream** with the [Close](#) method.
- Input bytes or text to a **Stream** with the [Write](#) and [WriteText](#) methods.
- Read bytes from the **Stream** with the [Read](#) and [ReadText](#) methods.
- Write any **Stream** data still in the ADO buffer to the underlying object with the [Flush](#) method.
- Copy the contents of a **Stream** to another **Stream** with the [CopyTo](#) method.
- Control how lines are read from the source file with the [SkipLine](#) method and the [LineSeparator](#) property.

- Determine the end of stream position with the [EOS](#) property and [SetEOS](#) method.
- Save and restore data in files with the [SaveToFile](#) and [LoadFromFile](#) methods.
- Specify the character set used for storing the **Stream** with the [Charset](#) property.
- Halt an asynchronous **Stream** operation with the [Cancel](#) method.
- Determine the number of bytes in a **Stream** with the [Size](#) property.
- Control the current position within a **Stream** with the [Position](#) property.
- Determine the type of data in a **Stream** with the [Type](#) property.
- Determine the current state of the **Stream** (closed, open, or executing) with the [State](#) property.
- Specify the access mode for the **Stream** with the [Mode](#) property.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Visual Basic Example](#)

[Stream Object Properties, Methods, and Events](#) | [Chapter 10: Records and Streams](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Stream Object Properties, Methods, and Events

Properties

[Charset Property](#)

[EOS Property](#)

[LineSeparator Property](#)

[Mode Property](#)

[Position Property](#)

[Size Property \(ADO Stream\)](#)

[State Property](#)

[Type Property \(ADO Stream\)](#)

Methods

[Cancel Method](#)

[Close Method](#)

[CopyTo Method](#)

[Flush Method](#)

[LoadFromFile Method](#)

[Open Method \(ADO Stream\)](#)

[Read Method](#)

[ReadText Method](#)

[SaveToFile Method](#)

[SetEOS Method](#)

[SkipLine Method](#)

[Stat Method](#)

[Write Method](#)

[WriteText Method](#)

Events

None.

See Also

[Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Collections

The relationships between these collections and the ADO objects are represented in the [ADO Object Model](#).

Each collection can contain its corresponding object. For example, an [Error](#) object can be contained in an [Errors](#) collection. For more information about objects, see [ADO Objects](#) or a specific object topic.

Errors	Contains all the Error objects created in response to a single provider-related failure.
Fields	Contains all the Field objects of a Recordset object.
Parameters	Contains all the Parameter objects of a Command object.
Properties	Contains all the Property objects for a specific instance of an object.

See Also

[ADO API Reference](#) | [ADO Dynamic Properties](#) | [ADO Enumerated Constants](#) | [Appendix B: DO Errors](#) | [ADO Events](#) | [ADO Methods](#) | [ADO Object Model](#) | [ADO Objects](#) | [ADO Properties](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Errors Collection

Contains all the [Error](#) objects created in response to a single provider-related failure [provider](#).

Connection



Remarks

Any operation involving ADO objects can generate one or more provider errors. As each error occurs, one or more **Error** objects can be placed in the **Errors** collection of the [Connection](#) object. When another ADO operation generates an error, the **Errors** collection is cleared, and the new set of **Error** objects can be placed in the **Errors** collection.

Each **Error** object represents a specific provider error, not an ADO error. ADO errors are exposed to the run-time exception-handling mechanism. For example, in Microsoft Visual Basic, the occurrence of an ADO-specific error will trigger an [onError](#) event and appear in the **Err** object.

ADO operations that don't generate an error have no effect on the **Errors** collection. Use the [Clear](#) method to manually clear the **Errors** collection.

The set of **Error** objects in the **Errors** collection describes all errors that occurred in response to a single statement. Enumerating the specific errors in the **Errors** collection enables your error-handling routines to more precisely determine the cause and origin of an error, and take appropriate steps to recover.

Some properties and methods return warnings that appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the [Resync](#), [UpdateBatch](#), or [CancelBatch](#) methods on a [Recordset](#) object, the [Open](#) method on a **Connection** object, or set the [Filter](#) property on a **Recordset** object, call the **Clear** method on the **Errors** collection. That way you can read the [Count](#) property of the **Errors** collection to test for returned warnings.

Note See the **Error** object topic for a more detailed explanation of the way a single ADO operation can generate multiple errors.

See Also

[Errors Collection Properties, Methods, and Events](#) | [Error Object](#) | [Appendix A: Providers](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Errors Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Clear Method](#)

[Refresh Method](#)

Events

None.

See Also

[Errors Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Fields Collection

Contains all the [Field](#) objects of a [Recordset](#) or [Record](#) object.

Recordset



Remarks

A **Recordset** object has a **Fields** collection made up of **Field** objects. Each **Field** object corresponds to a column in the **Recordset**. You can populate the **Fields** collection before opening the **Recordset** by calling the [Refresh](#) method on the collection.

Note See the **Field** object topic for a more detailed explanation of how to use **Field** objects.

The **Fields** collection has an [Append](#) method, which provisionally creates and adds a **Field** object to the collection, and an **Update** method, which finalizes any additions or deletions.

A **Record** object has two special fields that can be indexed with [FieldEnum](#) constants. One constant accesses a field containing the default stream for the **Record**, and the other accesses a field containing the absolute URL string for the **Record**.

Certain providers (for example, the [Microsoft OLE DB Provider for Internet Publishing](#)) may populate the **Fields** collection with a subset of available fields for the **Record** or **Recordset**. Other fields will not be added to the collection until they are first referenced by name or indexed by your code.

If you attempt to reference a nonexistent field by name, a new **Field** object will be appended to the **Fields** collection with a [Status](#) of **adFieldPendingInsert**. When you call [Update](#), ADO will create a new field in your data source if allowed by your provider.

See Also

[Fields Collection Properties, Methods, and Events](#) | [Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Fields Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method](#)

[CancelUpdate Method](#)

[Delete Method \(ADO Fields Collection\)](#)

[Refresh Method](#)

[Resync Method](#)

[Update Method](#)

Events

None.

See Also

[Fields Collection](#)

ADO 2.5 API Reference

Parameters Collection

Contains all the [Parameter](#) objects of a [Command](#) object.

Command



Remarks

A **Command** object has a **Parameters** collection made up of **Parameter** objects.

Using the [Refresh](#) method on a **Command** object's **Parameters** collection retrieves provider parameter information for the stored procedure or parameterized query specified in the **Command** object. Some [providers](#) do not support stored procedure calls or [parameterized](#) queries; calling the **Refresh** method on the **Parameters** collection when using such a provider will return an error.

If you have not defined your own **Parameter** objects and you access the **Parameters** collection before calling the **Refresh** method, ADO will automatically call the method and populate the collection for you.

You can minimize calls to the provider to improve performance if you know the properties of the parameters associated with the stored procedure or parameterized query you wish to call. Use the [CreateParameter](#) method to create **Parameter** objects with the appropriate property settings and use the [Append](#) method to add them to the **Parameters** collection. This lets you set and return parameter values without having to call the provider for the parameter information. If you are writing to a provider that does not supply parameter information, you must manually populate the **Parameters** collection using this method to be able to use parameters at all. Use the [Delete](#) method to remove **Parameter** objects from the **Parameters** collection if necessary.

The objects in the **Parameters** collection of a **Recordset** go out of scope (therefore becoming unavailable) when the **Recordset** is closed.

See Also

[Parameters Collection Properties, Methods, and Events](#) | [Append Method](#) | [CreateParameter Method](#) | [Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Parameters Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method](#)

[Delete Method \(ADO Parameters Collection\)](#)

[Refresh Method](#)

Events

None.

See Also

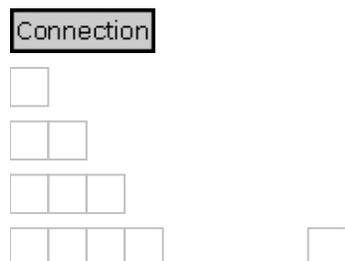
[Parameters Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Properties Collection

Contains all the [Property](#) objects for a specific instance of an object.



Remarks

Some ADO objects have a **Properties** collection made up of **Property** objects. Each **Property** object corresponds to a characteristic of the ADO object specific to the [provider](#).

Note See the [Property](#) object topic for a more detailed explanation of how to use **Property** objects.

The **Dynamic Properties** of the **Recordset** object go out of scope (become unavailable) when the **Recordset** is closed.

See Also

[Properties Collection Properties, Methods, and Events](#) | [Property Object](#) | [Appendix A: Providers](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Properties Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Refresh Method](#)

Events

None.

See Also

[Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Properties

AbsolutePage	Indicates on which page the current record resides.
AbsolutePosition	Indicates the ordinal position of a Recordset object's current record.
ActiveCommand	Indicates the Command object that created the associated Recordset object.
ActiveConnection	Indicates to which Connection object the specified Command , Recordset , or Record object currently belongs.
ActualSize	Indicates the actual length of a field's value.
Attributes	Indicates one or more characteristics of an object. BOF — indicates that the current record position is before the first record in a Recordset object.
BOF and EOF	EOF — indicates that the current record position is after the last record in a Recordset object.
Bookmark	Indicates a bookmark that uniquely identifies the current record in a Recordset object or sets the current record in a Recordset object to the record identified by a valid bookmark.
CacheSize	Indicates the number of records from a Recordset object that are cached locally in memory.
Chapter	Gets or sets an OLE DB Chapter object from/on an ADOREcordsetConstruction object.
CharSet	Indicates the character set into which the contents of a text Stream should be translated.
CommandText	Indicates the text of a command to be issued against a provider .
CommandTimeout	Indicates how long to wait while executing a command before terminating the attempt and generating an error.
CommandType	Indicates the type of a Command object.
ConnectionString	Indicates the information used to establish a connection

Property	to a data source.
ConnectionTimeout	Indicates how long to wait while establishing a connection before terminating the attempt and generating an error.
Count	Indicates the number of objects in a collection.
CursorLocation	Indicates the location of the cursor service.
CursorType	Indicates the type of cursor used in a Recordset object.
DataMember	Indicates the name of the data member that will be retrieved from the object referenced by the DataSource property.
DataSource	Indicates an object that contains data to be represented as a Recordset object.
DefaultDatabase	Indicates the default database for a Connection object.
DefinedSize	Indicates the data capacity of a Field object.
Description	Describes an Error object.
Direction	Indicates whether the Parameter represents an input parameter, an output parameter, or both, or if the parameter is the return value from a stored procedure.
EditMode	Indicates the editing status of the current record.
EOS	Indicates whether the current position is at the end of the stream.
Filter	Indicates a filter for data in a Recordset .
HelpContext and HelpFile	Indicates the help file and topic associated with an Error object. HelpContextID — returns a context ID, as a Long value, for a topic in a Help file. HelpFile — returns a String value that evaluates to a fully resolved path to a Help file.
Index	Indicates the name of the index currently in effect for a Recordset object.
IsolationLevel	Indicates the level of isolation for a Connection object.
Item	Indicates a specific member of a collection, by name or

	ordinal number.
<u>LineSeparator</u>	Indicates the binary character to be used as the line separator in text Stream objects.
<u>LockType</u>	Indicates the type of locks placed on records during editing.
<u>MarshalOptions</u>	Indicates which records are to be marshaled back to the server.
<u>MaxRecords</u>	Indicates the maximum number of records to return to a Recordset from a query.
<u>Mode</u>	Indicates the available permissions for modifying data in a Connection , Record , or Stream object.
<u>Name</u>	Indicates the name of an object.
<u>NativeError</u>	Indicates the provider-specific error code for a given Error object.
<u>Number</u>	Indicates the number that uniquely identifies an Error object.
<u>NumericScale</u>	Indicates the scale of numeric values in a Parameter or Field object.
<u>OriginalValue</u>	Indicates the value of a Field that existed in the record before any changes were made.
<u>PageCount</u>	Indicates how many pages of data the Recordset object contains.
<u>PageSize</u>	Indicates how many records constitute one page in the Recordset .
<u>ParentRow</u>	Sets the container of an OLE DB Row object on an ADORecordConstruction object, so that the parent of the row is turned into an ADO Record object.
<u>ParentURL</u>	Indicates an absolute URL string that points to the parent Record of the current Record object.
<u>Position</u>	Indicates the current position within a Stream object.
<u>Precision</u>	Indicates the degree of precision for numeric values in a Parameter object or for numeric Field objects.
<u>Prepared</u>	Indicates whether to save a compiled version of a command before execution.
<u>Provider</u>	Indicates the name of the provider for a Connection object.

<u>RecordCount</u>	Indicates the number of records in a Recordset object.
<u>RecordType</u>	Indicates the type of Record object.
<u>Row</u>	Gets or sets an OLE DB Row object from/on an ADORowConstruction object.
<u>RowPosition</u>	Gets or sets an OLE DB RowPosition object from/on an ADORowsetConstruction object.
<u>Rowset</u>	Gets or sets an OLE DB Rowset object from/on an ADORowsetConstruction object.
<u>Size</u>	Indicates the maximum size, in bytes or characters, of a Parameter object.
<u>Size (ADO Stream)</u>	Indicates the total size of the stream in number of bytes.
<u>Sort</u>	Indicates one or more field names on which the Recordset is sorted, and whether each field is sorted in ascending or descending order.
<u>Source (ADO Error)</u>	Indicates the name of the object or application that originally generated an error.
<u>Source (ADO Record)</u>	Indicates the entity represented by the Record object.
<u>Source (ADO Recordset)</u>	Indicates the source for the data in a Recordset object
<u>SQLState</u>	Indicates the SQL state for a given Error object. Indicates for all applicable objects whether the state of the object is open or closed.
<u>State</u>	Indicates for all applicable objects executing an asynchronous method, whether the current state of the object is connecting, executing, or retrieving
<u>Status (ADO Field)</u>	Indicates the status of a Field object.
<u>Status (ADO Recordset)</u>	Indicates the status of the current record with respect to batch updates or other bulk operations. Indicates, in a hierarchical Recordset object, whether the reference to the underlying child records (that is, the <i>chapter</i>) changes when the parent row position changes.
<u>StayInSync</u>	

Type	Indicates the operational type or data type of a Parameter , Field , or Property object.
Type (ADO Stream)	Indicates the type of data contained in the Stream (binary or text).
UnderlyingValue	Indicates a Field object's current value in the database.
Value	Indicates the value assigned to a Field , Parameter , or Property object.
Version	Indicates the ADO version number.

See Also

[ADO API Reference](#) | [ADO Collections](#) | [ADO Dynamic Properties](#) | [ADO Enumerated Constants](#) | [Appendix B: ADO Errors](#) | [ADO Events](#) | [ADO Methods](#) | [ADO Object Model](#) | [ADO Objects](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

AbsolutePage Property

Indicates on which page the current record resides.

Settings and Return Values

Sets or returns a **Long** value from 1 to the number of pages in the [Recordset](#) object ([PageCount](#)), or returns one of the [PositionEnum](#) values.

Remarks

This property can be used to identify the page number on which the current record is located. It uses the [PageSize](#) property to logically divide the total rowset count of the **Recordset** object into a series of pages, each of which has the number of records equal to **PageSize** (except for the last page, which may have fewer records). The [provider](#) must support the appropriate functionality for this property to be available.

When getting or setting the **AbsolutePage** property, ADO uses the [AbsolutePosition](#) property and the [PageSize](#) property together as follows:

- To get the **AbsolutePage**, ADO first retrieves the **AbsolutePosition**, and then divides it by the **PageSize**.
- To set the **AbsolutePage**, ADO moves the **AbsolutePosition** as follows: it multiplies the **PageSize** by the new **AbsolutePage** value and then adds 1 to the value. As a result, the current position in the **Recordset** after successfully setting **AbsolutePage** is, the first record in that page.

Like the **AbsolutePosition** property, **AbsolutePage** is 1-based and equals 1 when the current record is the first record in the **Recordset**. Set this property to move to the first record of a particular page. Obtain the total number of pages from the **PageCount** property.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AbsolutePosition Property](#) | [PageCount Property](#) | [PageSize Property](#)

Applies To: [Recordset Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

AbsolutePosition Property

Indicates the ordinal position of a [Recordset](#) object's current record.

Settings and Return Values

Sets or returns a **Long** value from 1 to the number of records in the **Recordset** object ([RecordCount](#)), or returns one of the [PositionEnum](#) values.

Remarks

In order to set the **AbsolutePosition** property, ADO requires that the OLE DB provider you are using implement the *IRowsetLocate* interface.

Accessing the **AbsolutePosition** property of a **Recordset** that was opened with either a forward-only or dynamic cursor raises the error **adErrFeatureNotAvailable**. With other cursor types, the correct position will be returned as long as the provider supports the *IRowsetScroll* interface. If the provider does not support the *IRowsetScroll* interface, the property is set to **adPosUnknown**. See the documentation for your provider to determine whether it supports *IRowsetScroll*.

Use the **AbsolutePosition** property to move to a record based on its ordinal position in the **Recordset** object, or to determine the ordinal position of the current record. The [provider](#) must support the appropriate functionality for this property to be available.

Like the [AbsolutePage](#) property, **AbsolutePosition** is 1-based and equals 1 when the current record is the first record in the **Recordset**. You can obtain the total number of records in the **Recordset** object from the [RecordCount](#) property.

When you set the **AbsolutePosition** property, even if it is to a record in the current cache, ADO reloads the cache with a new group of records starting with the record you specified. The [CacheSize](#) property determines the size of this group.

Note You should not use the **AbsolutePosition** property as a surrogate record number. The position of a given record changes when you delete a preceding record. There is also no assurance that a given record will have the same **AbsolutePosition** if the **Recordset** object is requested or reopened. Bookmarks are still the recommended way of retaining and returning to a given position and are the only way of positioning across all types of **Recordset** objects.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AbsolutePage Property](#) | [RecordCount Property](#)

Applies To: [Recordset Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ActiveCommand Property

Indicates the [Command](#) object that created the associated [Recordset](#) object.

Return Value

Returns a **VARIANT** that contains a **Command** object. Default is a null object reference.

Remarks

The **ActiveCommand** property is read-only.

If a **Command** object was not used to create the current **Recordset**, then a **Null** object reference is returned.

Use this property to find the associated **Command** object when you are given only the resulting **Recordset** object.

See Also

[Visual Basic Example](#) | [JScript Example](#) | [VC++Example](#) | [VJ++Example](#)

[Command Object](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ActiveConnection Property

Indicates to which [Connection](#) object the specified [Command](#), [Recordset](#), or [Record](#) object currently belongs.

Settings and Return Values

Sets or returns a **String** value that contains a definition for a connection if the connection is closed, or a **Variant** containing the current **Connection** object if the connection is open. Default is a null object reference. See the [ConnectionString](#) property.

Remarks

Use the **ActiveConnection** property to determine the **Connection** object over which the specified **Command** object will execute or the specified **Recordset** will be opened.

Command

For **Command** objects, the **ActiveConnection** property is read/write.

If you attempt to call the [Execute](#) method on a **Command** object before setting this property to an open **Connection** object or valid connection string, an error occurs.

Microsoft Visual Basic Setting the **ActiveConnection** property to *Nothing* disassociates the **Command** object from the current **Connection** and causes the [provider](#) to release any associated resources on the data source. You can then associate the **Command** object with the same or another **Connection** object. Some providers allow you to change the property setting from one **Connection** to another, without having to first set the property to *Nothing*.

If the [Parameters](#) collection of the **Command** object contains parameters supplied by the provider, the collection is cleared if you set the

ActiveConnection property to *Nothing* or to another **Connection** object. If you manually create [Parameter](#) objects and use them to fill the **Parameters** collection of the **Command** object, setting the **ActiveConnection** property to *Nothing* or to another **Connection** object leaves the **Parameters** collection intact.

Closing the **Connection** object with which a **Command** object is associated sets the **ActiveConnection** property to *Nothing*. Setting this property to a closed **Connection** object generates an error.

Recordset

For open **Recordset** objects or for **Recordset** objects whose [Source](#) property is set to a valid **Command** object, the **ActiveConnection** property is read-only. Otherwise, it is read/write.

You can set this property to a valid **Connection** object or to a valid connection string. In this case, the provider creates a new **Connection** object using this definition and opens the connection. Additionally, the provider may set this property to the new **Connection** object to give you a way to access the **Connection** object for extended error information or to execute other commands.

If you use the *ActiveConnection* argument of the [Open](#) method to open a **Recordset** object, the **ActiveConnection** property will inherit the value of the argument.

If you set the **Source** property of the **Recordset** object to a valid **Command** object variable, the **ActiveConnection** property of the **Recordset** inherits the setting of the **Command** object's **ActiveConnection** property.

Remote Data Service Usage When used on a [client-side Recordset](#) object, this property can be set only to a connection string or (in Microsoft Visual Basic or Visual Basic, Scripting Edition) to *Nothing*.

Record

This property is read/write when the **Record** object is closed, and may contain a connection string or reference to an open **Connection** object. This property is read-only when the **Record** object is open, and contains a reference to an open

Connection object.

A **Connection** object is created implicitly when the **Record** object is opened from a [URL](#). Open the **Record** with an existing, open **Connection** object by assigning the **Connection** object to this property, or using the **Connection** object as a parameter in the [Open](#) method call. If the **Record** is opened from an existing **Record** or [Recordset](#), then it is automatically associated with that **Record** or **Recordset** object's **Connection** object.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++Example](#) | [JScriptExample](#)

[Connection Object](#) | [ConnectionString Property](#)

Applies To: [Command Object](#) | [Record Object](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ActualSize Property

Indicates the actual length of a field's value.

Settings and Return Values

Returns a **Long** value. Some providers may allow this property to be set to reserve space for BLOB data, in which case the default value is 0.

Remarks

Use the **ActualSize** property to return the actual length of a [Field](#) object's value. For all fields, the **ActualSize** property is read-only. If ADO cannot determine the length of the **Field** object's value, the **ActualSize** property returns **adUnknown**.

The **ActualSize** and [DefinedSize](#) properties are different, as shown in the following example. A **Field** object with a declared type of **adVarChar** and a maximum length of 50 characters returns a **DefinedSize** property value of 50, but the **ActualSize** property value it returns is the length of the data stored in the field for the current record. **Fields** with a **DefinedSize** greater than 255 bytes are treated as variable length columns.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[DefinedSize Property](#)

Applies To: [Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Attributes Property

Indicates one or more characteristics of an object.

Settings and Return Values

Sets or returns a **Long** value.

For a [Connection](#) object, the **Attributes** property is read/write, and its value can be the sum of one or more [XactAttributeEnum](#) values. The default is zero (0).

For a [Parameter](#) object, the **Attributes** property is read/write, and its value can be the sum of any one or more [ParameterAttributesEnum](#) values. The default is **adParamSigned**.

For a [Field](#) object, the **Attributes** property can be the sum of one or more [FieldAttributeEnum](#) values. It is normally read-only, However, for new **Field** objects that have been appended to the [Fields](#) collection of a [Record](#), **Attributes** is read/write only after the [Value](#) property for the **Field** has been specified and the new **Field** has been successfully added by the data provider by calling the [Update](#) method of the **Fields** collection.

For a [Property](#) object, the **Attributes** property is read-only, and its value can be the sum of any one or more [PropertyAttributesEnum](#) values.

Remarks

Use the **Attributes** property to set or return characteristics of **Connection** objects, **Parameter** objects, [Field](#) objects, or [Property](#) objects.

When you set multiple attributes, you can sum the appropriate constants. If you set the property value to a sum including incompatible constants, an error occurs.

Remote Data Service Usage This property is not available on a client-side **Connection** object.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AppendChunk Method](#) | [BeginTrans, CommitTrans, and RollbackTrans Methods](#)
| [GetChunk Method](#)

Applies To: [Connection Object](#) | [Field Object](#) | [Parameter Object](#) | [Property Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

BOF, EOF Properties

- **BOF** — Indicates that the current record position is before the first record in a [Recordset](#) object.
- **EOF** — Indicates that the current record position is after the last record in a **Recordset** object.

Return Value

The **BOF** and **EOF** properties return **Boolean** values.

Remarks

Use the **BOF** and **EOF** properties to determine whether a **Recordset** object contains records or whether you've gone beyond the limits of a **Recordset** object when you move from record to record.

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

If either the **BOF** or **EOF** property is **True**, there is no current record.

If you open a **Recordset** object containing no records, the **BOF** and **EOF** properties are set to **True** (see the [RecordCount](#) property for more information about this state of a **Recordset**). When you open a **Recordset** object that contains at least one record, the first record is the current record and the **BOF** and **EOF** properties are **False**.

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.

This table shows which **Move** methods are allowed with different combinations of the **BOF** and **EOF** properties.

	MoveFirst, MoveLast	MovePrevious, Move < 0	Move 0	MoveNext, Move > 0
BOF=True, EOF=False	Allowed	Error	Error	Allowed
BOF=False, EOF=True	Allowed	Allowed	Error	Error
Both True	Error	Error	Error	Error
Both False	Allowed	Allowed	Allowed	Allowed

Allowing a **Move** method doesn't guarantee that the method will successfully locate a record; it only means that calling the specified **Move** method won't generate an error.

The following table shows what happens to the **BOF** and **EOF** property settings when you call various **Move** methods but are unable to successfully locate a record.

	BOF	EOF
MoveFirst, MoveLast	Set to True	Set to True
Move 0	No change	No change
MovePrevious, Move < 0	Set to True	No change
MoveNext, Move > 0	No change	Set to True

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Bookmark Property

Indicates a bookmark that uniquely identifies the current record in a [Recordset](#) object or sets the current record in a **Recordset** object to the record identified by a valid bookmark.

Settings and Return Values

Sets or returns a **Variant** expression that evaluates to a valid bookmark.

Remarks

Use the **Bookmark** property to save the position of the current record and return to that record at any time. Bookmarks are available only in **Recordset** objects that support bookmark functionality.

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 quickly return to that record at any time after moving to a different record, set the **Recordset** object's **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 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.

Remote Data Service Usage When used on a client-side **Recordset** object, the **Bookmark** property is always available.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Supports Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CacheSize Property

Indicates the number of records from a [Recordset](#) object that are cached locally in memory.

Settings and Return Values

Sets or returns a **Long** value that must be greater than 0. Default is 1.

Remarks

Use the **CacheSize** property to control how many records to retrieve at one time into local memory from the [provider](#). For example, if the **CacheSize** is 10, after first opening the **Recordset** object, the provider retrieves the first 10 records into local memory. As you move through the **Recordset** object, the provider returns the data from the local memory buffer. As soon as you move past the last record in the cache, the provider retrieves the next 10 records from the data source into the cache.

Note **CacheSize** is based on the **Maximum Open Rows** provider-specific property (in the **Properties** collection of the **Recordset** object). You cannot set **CacheSize** to a value greater than **Maximum Open Rows**. To modify the number of rows which can be opened by the provider, set **Maximum Open Rows**.

The value of **CacheSize** can be adjusted during the life of the **Recordset** object, but changing this value only affects the number of records in the cache after subsequent retrievals from the data source. Changing the property value alone will not change the current contents of the cache.

If there are fewer records to retrieve than **CacheSize** specifies, the provider returns the remaining records and no error occurs.

A **CacheSize** setting of zero is not allowed and returns an error.

Records retrieved from the cache don't reflect concurrent changes that other

users made to the source data. To force an update of all the cached data, use the [Resync](#) method.

If **CacheSize** is set to a value greater than one, the navigation methods ([Move](#), [MoveFirst](#), [MoveLast](#), [MoveNext](#), and [MovePrevious](#)) may result in navigation to a deleted record, if deletion occurs after the records were retrieved. After the initial fetch, subsequent deletions will not be reflected in your data cache until you attempt to access a data value from a deleted row. However, setting **CacheSize** to one eliminates this issue since deleted rows cannot be fetched.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#) | [Jscript Example](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Chapter Property

Gets or sets an OLE DB **Chapter** object from/on an **ADORecordsetConstruction** object. When you use **put_Chapter** to set the **Chapter** object, a subset of rows is turned into an ADO **Recordset** object. This sets the current chapter of the **Rowset** object.

Read/write.Syntax

```
HRESULT get_Chapter([out, retval] long* plChapter);  
HRESULT put_Chapter([in] long lChapter);
```

Parameters

plChapter

Pointer to the handle of a chapter.

LChapter

Handle of a chapter.

Return Values

This property method returns the standard HRESULT values, including S_OK and E_FAIL.

Applies To

[ADORecordsetConstruction](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Charset Property

Indicates the character set into which the contents of a text [Stream](#) should be translated for storage in the Stream objects internal buffer.

Settings and Return Values

Sets or returns a **String** value that specifies the character set into which the contents of the **Stream** will be translated. The default value is "Unicode". Allowed values are typical strings passed over the interface as Internet character set strings (for example, "iso-8859-1", "Windows-1252", etc.). For a list of the character set strings that is known by a system, see the subkeys of HKEY_CLASSES_ROOT\MIME\Database\Charset in the Windows Registry.

Remarks

In a text **Stream** object, text data is stored in the character set specified by the **Charset** property. The default is Unicode. The **Charset** property is used for converting data going into the **Stream** or coming out of the **Stream**. For example, if the **Stream** contains ISO-8859-1 data and that data is copied to a BSTR, the **Stream** object will convert the data to Unicode. The reverse is also true.

For an open **Stream**, the current [Position](#) must be at the beginning of the **Stream** (0) to be able to set **Charset**.

Charset is used only with text **Stream** objects ([Type](#) is **adTypeText**). This property is ignored if **Type** is **adTypeBinary**.

See Also

[Visual Basic Example](#)

Applies To: [Stream Object](#)

ADO 2.5 API Reference

CommandText Property

Indicates the text of a command to be issued against a [provider](#).

Settings and Return Values

Sets or returns a **String** value that contains a provider command, such as an SQL statement, a table name, a [relative URL](#), or a stored procedure call. Default is "" (zero-length string).

Remarks

Use the **CommandText** property to set or return the text of a command represented by a [Command](#) object. Usually this will be an SQL statement, but can also be any other type of command statement recognized by the provider, such as a stored procedure call. An SQL statement must be of the particular dialect or version supported by the provider's query processor.

If the [Prepared](#) property of the **Command** object is set to **True** and the **Command** object is bound to an open connection when you set the **CommandText** property, ADO prepares the query (that is, a compiled form of the query that is stored by the provider) when you call the [Execute](#) or **Open** methods.

Depending on the [CommandType](#) property setting, ADO may alter the **CommandText** property. You can read the **CommandText** property at any time to see the actual command text that ADO will use during execution.

Use the **CommandText** property to set or return a relative URL that specifies a resource, such as a file or directory. The resource is relative to a location specified explicitly by an absolute URL, or implicitly by an open [Connection](#) object.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Requery Method](#) | [JScriptExample](#)

Applies To: [Command Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CommandTimeout Property

Indicates how long to wait while executing a command before terminating the attempt and generating an error.

Settings and Return Values

Sets or returns a **Long** value that indicates, in seconds, how long to wait for a command to execute. Default is 30.

Remarks

Use the **CommandTimeout** property on a [Connection](#) object or [Command](#) object to allow the cancellation of an [Execute](#) method call, due to delays from network traffic or heavy server use. If the interval set in the **CommandTimeout** property elapses before the command completes execution, an error occurs and ADO cancels the command. If you set the property to zero, ADO will wait indefinitely until the execution is complete. Make sure the [provider](#) and data source to which you are writing code support the **CommandTimeout** functionality.

The **CommandTimeout** setting on a **Connection** object has no effect on the **CommandTimeout** setting on a **Command** object on the same **Connection**; that is, the **Command** object's **CommandTimeout** property does not inherit the value of the **Connection** object's **CommandTimeout** value.

On a **Connection** object, the **CommandTimeout** property remains read/write after the **Connection** is opened.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#) | [JScriptExample](#)

[ConnectionTimeout Property](#)

Applies To: [Command Object](#) | [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CommandType Property

Indicates the type of a [Command](#) object.

Settings and Return Values

Sets or returns one or more [CommandTypeEnum](#) values.

Note Do not use the **CommandTypeEnum** values of **adCmdFile** or **adCmdTableDirect** with **CommandType**. These values can only be used as options with the [Open](#) and [Requery](#) methods of a [Recordset](#).

Remarks

Use the **CommandType** property to optimize evaluation of the [CommandText](#) property.

If the **CommandType** property value equals **adCmdUnknown** (the default value), you may experience diminished performance because ADO must make calls to the [provider](#) to determine if the **CommandText** property is an SQL statement, a stored procedure, or a table name. If you know what type of command you're using, setting the **CommandType** property instructs ADO to go directly to the relevant code. If the **CommandType** property does not match the type of command in the **CommandText** property, an error occurs when you call the [Execute](#) method.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#) | [JScriptExample](#)

Applies To: [Command Object](#)

ADO 2.5 API Reference

ConnectionString Property

Indicates the information used to establish a connection to a data source.

Settings and Return Values

Sets or returns a **String** value.

Remarks

Use the **ConnectionString** property to specify a data source by passing a detailed connection string containing a series of *argument = value* statements separated by semicolons.

ADO supports five arguments for the **ConnectionString** property; any other arguments pass directly to the [provider](#) without any processing by ADO. The arguments ADO supports are as follows.

Argument	Description
<i>Provider=</i>	Specifies the name of a provider to use for the connection.
<i>File Name=</i>	Specifies the name of a provider-specific file (for example, a persisted data source object) containing preset connection information.
<i>Remote Provider=</i>	Specifies the name of a provider to use when opening a client-side connection. (Remote Data Service only.)
<i>Remote Server=</i>	Specifies the path name of the server to use when opening a client-side connection. (Remote Data Service only.)
<i>URL=</i>	Specifies the connection string as an absolute URL identifying a resource, such as a file or directory.

After you set the **ConnectionString** property and open the [Connection](#) object, the provider may alter the contents of the property, for example, by mapping the ADO-defined argument names to their provider equivalents.

The **ConnectionString** property automatically inherits the value used for the

ConnectionString argument of the [Open](#) method, so you can override the current **ConnectionString** property during the **Open** method call.

Because the *File Name* argument causes ADO to load the associated provider, you cannot pass both the *Provider* and *File Name* arguments.

The **ConnectionString** property is read/write when the connection is closed and read-only when it is open.

Duplicates of an argument in the **ConnectionString** property are ignored. The last instance of any argument is used.

Remote Data Service Usage When used on a client-side **Connection** object, the **ConnectionString** property can include only the *Remote Provider* and *Remote Server* parameters.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Appendix A: Providers](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ConnectionTimeout Property

Indicates how long to wait while establishing a connection before terminating the attempt and generating an error.

Settings and Return Values

Sets or returns a **Long** value that indicates, in seconds, how long to wait for the connection to open. Default is 15.

Remarks

Use the **ConnectionTimeout** property on a [Connection](#) object if delays from network traffic or heavy server use make it necessary to abandon a connection attempt. If the time from the **ConnectionTimeout** property setting elapses prior to the opening of the connection, an error occurs and ADO cancels the attempt. If you set the property to zero, ADO will wait indefinitely until the connection is opened. Make sure the [provider](#) to which you are writing code supports the **ConnectionTimeout** functionality.

The **ConnectionTimeout** property is read/write when the connection is closed and read-only when it is open.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[CommandTimeout Property](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Count Property

Indicates the number of objects in a collection.

Return Value

Returns a **Long** value.

Remarks

Use the **Count** property to determine how many objects are in a given collection.

Because numbering for members of a collection begins with zero, you should always code loops starting with the zero member and ending with the value of the **Count** property minus 1. If you are using Microsoft Visual Basic and want to loop through the members of a collection without checking the **Count** property, use the **For Each...Next** command.

If the **Count** property is zero, there are no objects in the collection.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Refresh Method](#)

Applies To: [Axes Collection](#) | [Columns Collection](#) | [CubeDefs Collection](#) | [Dimensions Collection](#) | [Errors Collection](#) | [Fields Collection](#) | [Groups Collection](#) | [Hierarchies Collection](#) | [Indexes Collection](#) | [Keys Collection](#) | [Levels Collection](#) | [Members Collection](#) | [Parameters Collection](#) | [Positions Collection](#) | [Procedures Collection](#) | [Properties Collection](#) | [Tables Collection](#) | [Users Collection](#) | [Views Collection](#)

ADO 2.5 API Reference

CursorLocation Property

Indicates the location of the [cursor](#) service.

Settings And Return Values

Sets or returns a **Long** value that can be set to one of the [CursorLocationEnum](#) values.

Remarks

This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server.

This property setting affects connections established only after the property has been set. Changing the **CursorLocation** property has no effect on existing connections.

Cursors returned by the [Execute](#) method inherit this setting. **Recordset** objects will automatically inherit this setting from their associated connections.

This property is read/write on a [Connection](#) or a closed [Recordset](#), and read-only on an open **Recordset**.

Remote Data Service Usage When used on a client-side **Recordset** or **Connection** object, the **CursorLocation** property can only be set to **adUseClient**.

See Also

[Appendix A: Providers](#)

Applies To: [Connection Object](#) | [Recordset Object](#)

ADO 2.5 API Reference

CursorType Property

Indicates the type of [cursor](#) used in a [Recordset](#) object.

Settings and Return Values

Sets or returns a [CursorTypeEnum](#) value. The default value is **adOpenForwardOnly**.

Remarks

Use the **CursorType** property to specify the type of cursor that should be used when opening the **Recordset** object.

Only a setting of **adOpenStatic** is supported if the [CursorLocation](#) property is set to **adUseClient**. If an unsupported value is set, then no error will result; the closest supported **CursorType** will be used instead.

If a [provider](#) does not support the requested cursor type, it may return another cursor type. The **CursorType** property will change to match the actual cursor type in use when the [Recordset](#) object is open. To verify specific functionality of the returned cursor, use the [Supports](#) method. After you close the **Recordset**, the **CursorType** property reverts to its original setting.

The following chart shows the provider functionality (identified by **Supports** method constants) required for each cursor type.

For a Recordset of this CursorType	The Supports method must return True for all of these constants
adOpenForwardOnly	none
adOpenKeyset	adBookmark, adHoldRecords, adMovePrevious, adResync
adOpenDynamic	adMovePrevious
adOpenStatic	adBookmark, adHoldRecords, adMovePrevious, adResync

Note Although **Supports(adUpdateBatch)** may be true for dynamic and forward-only cursors, for batch updates you should use either a keyset or static cursor. Set the [LockType](#) property to **adLockBatchOptimistic** and the **CursorLocation** property to **adUseClient** to enable the Cursor Service for OLE DB, which is required for batch updates.

The **CursorType** property is read/write when the **Recordset** is closed and read-only when it is open.

Remote Data Service Usage When used on a [client-side Recordset](#) object, the **CursorType** property can be set only to **adOpenStatic**.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Supports Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

DataMember Property

Indicates the name of the data member that will be retrieved from the object referenced by the [DataSource](#) property.

Settings and Return Values

Sets or returns a **String** value. The name is not case sensitive.

Remarks

This property is used to create data-bound controls with the Data Environment. The Data Environment maintains collections of data (data sources) containing named objects (*data members*) that will be represented as a [Recordset](#) object.

The **DataMember** and **DataSource** properties must be used in conjunction.

The **DataMember** property determines which object specified by the **DataSource** property will be represented as a **Recordset** object. The **Recordset** object must be closed before this property is set. An error is generated if the **DataMember** property isn't set before the **DataSource** property, or if the **DataMember** name isn't recognized by the object specified in the **DataSource** property.

Usage

```
Dim rs as New ADODB.Recordset
rs.DataMember = "Command"      'Name of the rowset to bind to
Set rs.DataSource = myDE      'Name of the object containing an IRow
```

See Also

[DataSource Property](#)

Applies To: [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

DataSource Property

Indicates an object that contains data to be represented as a [Recordset](#) object.

Remarks

This property is used to create data-bound controls with the Data Environment. The Data Environment maintains collections of data (data sources) containing named objects (*data members*) that will be represented as a **Recordset** object.

The [DataMember](#) and **DataSource** properties must be used in conjunction.

The object referenced must implement the **IDataSource** interface and must contain an **IRowset** interface.

Usage

```
Dim rs as New ADODB.Recordset
rs.DataMember = "Command"      'Name of the rowset to bind to
Set rs.DataSource = myDE      'Name of the object containing an IRow
```

See Also

[DataMember Property](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

DefaultDatabase Property

Indicates the default database for a [Connection](#) object.

Settings and Return Values

Sets or returns a **String** value that evaluates to the name of a database available from the [provider](#).

Remarks

Use the **DefaultDatabase** property to set or return the name of the default database on a specific **Connection** object.

If there is a default database, SQL strings may use an unqualified syntax to access objects in that database. To access objects in a database other than the one specified in the **DefaultDatabase** property, you must qualify object names with the desired database name. Upon connection, the provider will write default database information to the **DefaultDatabase** property. Some providers allow only one database per connection, in which case you cannot change the **DefaultDatabase** property.

Some data sources and providers may not support this feature, and may return an error or an empty string.

Remote Data Service Usage This property is not available on a client-side **Connection** object.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

DefinedSize Property

Indicates the data capacity of a [Field](#) object.

Return Value

Returns a **Long** value that reflects the defined size of a field as a number of bytes.

Remarks

Use the **DefinedSize** property to determine the data capacity of a **Field** object.

The **DefinedSize** and [ActualSize](#) properties are different. For example, consider a **Field** object with a declared type of **adVarChar** and a **DefinedSize** property value of 50, containing a single character. The **ActualSize** property value it returns is the length in bytes of the single character.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[ActualSize Property](#)

Applies To: [Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Description Property

Describes an [Error](#) object.

Return Value

Returns a **String** value that contains a description of the error.

Remarks

Use the **Description** property to obtain a short description of the error. Display this property to alert the user to an error that you cannot or do not want to handle. The string will come from either ADO or a [provider](#).

Providers are responsible for passing specific error text to ADO. ADO adds an [Error](#) object to the **Errors** collection for each provider error or warning it receives. Enumerate the **Errors** collection to trace the errors that the provider passes.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[HelpContext, HelpFile Properties](#) | [Number Property](#) | [Source Property \(ADO Error\)](#)

Applies To: [Error Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Direction Property

Indicates whether the [Parameter](#) represents an input parameter, an output parameter, an input and an output parameter, or if the parameter is the return value from a stored procedure.

Settings and Return Values

Sets or returns a [ParameterDirectionEnum](#) value.

Remarks

Use the **Direction** property to specify how a parameter is passed to or from a procedure. The **Direction** property is read/write; this allows you to work with [providers](#) that don't return this information or to set this information when you don't want ADO to make an extra call to the provider to retrieve parameter information.

Not all providers can determine the direction of parameters in their stored procedures. In these cases, you must set the **Direction** property before you execute the query.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#) | [JScriptExample](#)

Applies To: [Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

EditMode Property

Indicates the editing status of the current record.

Return Value

Returns an [EditModeEnum](#) value.

Remarks

ADO maintains an editing buffer associated with the current record. This property indicates whether changes have been made to this buffer, or whether a new record has been created. Use the **EditMode** property to determine the editing status of the current record. You can test for pending changes if an editing process has been interrupted and determine whether you need to use the [Update](#) or [CancelUpdate](#) method.

See the [AddNew](#) method for a more detailed description of the **EditMode** property under different editing conditions.

When a call to [Delete](#) does not successfully delete the record or records in the data source (due to referential integrity violations, for example), the [Recordset](#) will remain in edit mode (**EditMode** = **adEditInProgress**). This means that **CancelUpdate** must be called before moving off the current record (with [Move](#), [NextRecordset](#), or [Close](#), for example).

Note EditMode can return a valid value only if there is a current record. **EditMode** will return an error if [BOF or EOF](#) is true, or if the current record has been deleted.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AddNew Method](#) | [CancelUpdate Method](#) | [Update Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

EOS Property

Indicates whether the current position is at the end of the stream.

Return Values

Returns a **Boolean** value that indicates whether the current position is at the end of the stream. **EOS** returns **True** if there are no more bytes in the stream; it returns **False** if there are more bytes following the current position.

To set the end of stream position, use the [SetEOS](#) method. To determine the current position, use the [Position](#) property.

See Also

[Visual Basic Example](#)

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Filter Property

Indicates a filter for data in a [Recordset](#).

Settings and Return Values

Sets or returns a **VARIANT** value, which can contain one of the following:

- **Criteria string** — a string made up of one or more individual clauses concatenated with **AND** or **OR** operators.
- **Array of bookmarks** — an array of unique bookmark values that point to records in the **Recordset** object.
- A [FilterGroupEnum](#) value.

Remarks

Use the **Filter** property to selectively screen out records in a **Recordset** object. The filtered **Recordset** becomes the current [cursor](#). Other properties that return values based on the current cursor are affected, such as [AbsolutePosition](#), [AbsolutePage](#), [RecordCount](#), and [PageCount](#). This is because setting the **Filter** property to a specific value will move the current record to the first record that satisfies the new value.

The criteria string is made up of clauses in the form *FieldName-Operator-Value* (for example, "LastName = 'Smith'"). You can create compound clauses by concatenating individual clauses with **AND** (for example, "LastName = 'Smith' AND FirstName = 'John'") or **OR** (for example, "LastName = 'Smith' OR LastName = 'Jones'"). Use the following guidelines for criteria strings:

- *FieldName* must be a valid field name from the **Recordset**. If the field name contains spaces, you must enclose the name in square brackets.
- *Operator* must be one of the following: <, >, <=, >=, <>, =, or **LIKE**.
- *Value* is the value with which you will compare the field values (for example, 'Smith', #8/24/95#, 12.345, or \$50.00). Use single quotes with strings and pound signs (#) with dates. For numbers, you can use decimal points, dollar signs, and scientific notation. If *Operator* is **LIKE**, *Value* can

use wildcards. Only the asterisk (*) and percent sign (%) wild cards are allowed, and they must be the last character in the string. *Value* cannot be null.

Note To include single quotation marks (') in the filter Value, use two single quotation marks to represent one. For example, to filter on O'Malley, the criteria string should be "col1 = 'O''Malley'". To include single quotation marks at both the beginning and the end of the filter value, enclose the string with pound signs (#). For example, to filter on '1', the criteria string should be "col1 = #'1'#".

- There is no precedence between **AND** and **OR**. Clauses can be grouped within parentheses. However, you cannot group clauses joined by an **OR** and then join the group to another clause with an **AND**, like this:

```
(LastName = 'Smith' OR LastName = 'Jones') AND FirstName = 'John'
```

- Instead, you would construct this filter as

```
(LastName = 'Smith' AND FirstName = 'John') OR (LastName = 'Jones' AND FirstName = 'John')
```

- In a **LIKE** clause, you can use a wildcard at the beginning and end of the pattern (for example, `LastName Like '*mit*'`), or only at the end of the pattern (for example, `LastName Like 'Smit*'`).

The filter constants make it easier to resolve individual record conflicts during batch update mode by allowing you to view, for example, only those records that were affected during the last [UpdateBatch](#) method call.

Setting the **Filter** property itself may fail because of a conflict with the underlying data (for example, a record has already been deleted by another user). In such a case, the [provider](#) returns warnings to the [Errors](#) collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. Use the [Status](#) property to locate records with conflicts.

Setting the **Filter** property to a zero-length string ("") has the same effect as using the **adFilterNone** constant.

Whenever the **Filter** property is set, the current record position moves to the first record in the filtered subset of records in the **Recordset**. Similarly, when the **Filter** property is cleared, the current record position moves to the first record in

the **Recordset**.

See the [Bookmark](#) property for an explanation of bookmark values from which you can build an array to use with the **Filter** property.

Only **Filters** in the form of Criteria Strings (e.g. OrderDate > '12/31/1999') affect the contents of a persisted **Recordset**. **Filters** created with an Array of **Bookmarks** or using a value from the **FilterGroupEnum** will not affect the contents of the persisted Recordset. These rules apply to **Recordsets** created with either client-side or server-side cursors.

Note When you apply the **adFilterPendingRecords** flag to a filtered and modified **Recordset** in the batch update mode, the resultant **Recordset** is empty if the filtering was based on the key field of a single-keyed table and the modification was made on the key field values. The resultant **Recordset** will be non-empty if one of the following is true:

- The filtering was based on non-key fields in a single-keyed table.
- The filtering was based on any fields in a multiple-keyed table.
- Modifications were made on non-key fields in a single-keyed table.
- Modifications were made on any fields in a multiple-keyed table.

The following table summarizes the effects of **adFilterPendingRecords** in different combinations of filtering and modifications. The left column shows the possible modifications; modifications can be made on any of the non-keyed fields, on the key field in a single-keyed table, or on any of the key fields in a multiple-keyed table. The top row shows the filtering criterion; filtering can be based on any of the non-keyed fields, the key field in a single-keyed table, or any of the key fields in a multiple-keyed table. The intersecting cells show the results: + means that applying **adFilterPendingRecords** results in a non-empty **Recordset**; - means an empty **Recordset**.

	Non keys	Single Key	Multiple Keys
Non keys	+	+	+
Single Key	+	-	N/A
Multiple Keys	+	N/A	+

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Clear Method](#) | [Optimize Property—Dynamic \(ADO\)](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

HelpContext, HelpFile Properties

Indicates the help file and topic associated with an [Error](#) object.

Return Values

- **HelpContextID** — returns a context ID, as a **Long** value, for a topic in a Help file.
- **HelpFile** — returns a **String** value that evaluates to a fully resolved path to a Help file.

Remarks

If a Help file is specified in the **HelpFile** property, the **HelpContext** property is used to automatically display the Help topic it identifies. If there is no relevant help topic available, the **HelpContext** property returns zero and the **HelpFile** property returns a zero-length string ("").

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Description Property](#) | [Number Property](#) | [Source Property \(ADO Error\)](#)

Applies To: [Error Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Index Property

Indicates the name of the index currently in effect for a [Recordset](#) object.

Settings and Return Values

Sets or returns a **String** value, which is the name of the index.

Remarks

The index named by the **Index** property must have previously been declared on the base table underlying the **Recordset** object. That is, the index must have been declared programmatically either as an ADOX [Index](#) object, or when the base table was created.

A run-time error will occur if the index cannot be set. The **Index** property cannot be set:

- Within a [WillChangeRecordset](#) or **RecordsetChangeComplete** event handler.
- If the **Recordset** is still executing an operation (which can be determined by the [State](#) property).
- If a filter has been set on the **Recordset** with the [Filter](#) property.

The **Index** property can always be set successfully if the **Recordset** is closed, but the **Recordset** will not open successfully, or the index will not be usable, if the underlying provider does not support indexes.

If the index can be set, the current row position may change. This will cause an update to the [AbsolutePosition](#) property, and the generation of **WillChangeRecordset**, **RecordsetChangeComplete**, [WillMove](#), and [MoveComplete](#) events.

If the index can be set and the [LockType](#) property is **adLockPessimistic** or **adLockOptimistic**, then an implicit [UpdateBatch](#) operation is performed. This releases the current and affected groups. Any existing filter is released, and the

current row position is changed to the first row of the reordered **Recordset**.

The **Index** property is used in conjunction with the [Seek](#) method. If the underlying [provider](#) does not support the **Index** property, and thus the **Seek** method, consider using the [Find](#) method instead. Determine whether the **Recordset** object supports indexes with the [Supports \(adIndex\)](#) method.

The built-in **Index** property is not related to the [dynamic Optimize](#) property, although they both deal with indexes.

See Also

[Visual Basic Example](#)

[Index Object](#) | [Seek Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

IsolationLevel Property

Indicates the level of isolation for a [Connection](#) object.

Settings and Return Values

Sets or returns an [IsolationLevelEnum](#) value. The default is **adXactChaos**.

Remarks

Use the **IsolationLevel** property to set the isolation level of a **Connection** object. The setting does not take effect until the next time you call the [BeginTrans](#) method. If the level of isolation you request is unavailable, the [provider](#) may return the next greater level of isolation.

The **IsolationLevel** property is read/write.

Remote Data Service Usage When used on a [client-side Connection](#) object, the **IsolationLevel** property can be set only to **adXactUnspecified**.

Because users are working with disconnected **Recordset** objects on a [client-side](#) cache, there may be multiuser issues. For instance, when two different users try to update the same record, Remote Data Service simply allows the user who updates the record first to "win." The second user's update request will fail with an error.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Item Property

Indicates a specific member of a collection, by name or ordinal number.

Syntax

```
Set object = collection.Item ( Index )
```

Return Value

Returns an object reference.

Parameters

Index

A **Variant** expression that evaluates either to the name or to the ordinal number of an object in a collection.

Remarks

Use the **Item** property to return a specific object in a collection. If **Item** cannot find an object in the collection corresponding to the *Index* argument, an error occurs. Also, some collections don't support named objects; for these collections, you must use ordinal number references.

The **Item** property is the default property for all collections; therefore, the following syntax forms are interchangeable:

```
collection.Item (Index)  
collection (Index)
```

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Axes Collection](#) | [Columns Collection](#) | [CubeDefs Collection](#) |

[Dimensions Collection](#) | [Errors Collection](#) | [Fields Collection](#) | [Groups Collection](#)
| [Hierarchies Collection](#) | [Indexes Collection](#) | [Keys Collection](#) | [Levels
Collection](#) | [Members Collection](#) | [Parameters Collection](#) | [Positions Collection](#) |
[Procedures Collection](#) | [Properties Collection](#) | [Tables Collection](#) | [Users
Collection](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

LineSeparator Property

Indicates the binary character to be used as the line separator in text [Stream](#) objects.

Settings and Return Values

Sets or returns a [LineSeparatorsEnum](#) value that indicates the line separator character used in the **Stream**. The default value is **adCRLF**.

Remarks

LineSeparator is used to interpret lines when reading the content of a text **Stream**. Lines can be skipped with the [SkipLine](#) method.

LineSeparator is used only with text **Stream** objects ([Type](#) is **adTypeText**). This property is ignored if **Type** is **adTypeBinary**.

See Also

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

LockType Property

Indicates the type of locks placed on records during editing.

Settings and Return Values

Sets or returns a [LockTypeEnum](#) value. The default value is **adLockReadOnly**.

Remarks

Set the **LockType** property before opening a [Recordset](#) to specify what type of locking the provider should use when opening it. Read the property to return the type of locking in use on an open **Recordset** object.

Providers may not support all lock types. If a provider cannot support the requested **LockType** setting, it will substitute another type of locking. To determine the actual locking functionality available in a **Recordset** object, use the [Supports](#) method with **adUpdate** and **adUpdateBatch**.

The **adLockPessimistic** setting is not supported if the [CursorLocation](#) property is set to **adUseClient**. If an unsupported value is set, then no error will result; the closest supported **LockType** will be used instead.

The **LockType** property is read/write when the **Recordset** is closed and read-only when it is open.

Remote Data Service Usage When used on a [client-side Recordset](#) object, the **LockType** property can only be set to **adLockBatchOptimistic**.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[CancelBatch Method](#) | [UpdateBatch Method](#)

Applies To: [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

MarshalOptions Property

Indicates which records are to be marshaled back to the server.

Settings And Return Values

Sets or returns a [MarshalOptionsEnum](#) value. The default value is **adMarshalAll**.

Remarks

When using a [client-side Recordset](#), records that have been modified on the client are written back to the [middle tier](#) or [Web server](#) through a technique called [marshaling](#), the process of packaging and sending interface method parameters across thread or process boundaries. Setting the **MarshalOptions** property can improve performance when modified remote data is marshaled for updating back to the middle tier or Web server.

Remote Data Service Usage This property is used only on a client-side **Recordset**.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

MaxRecords Property

Indicates the maximum number of records to return to a [Recordset](#) from a query.

Settings and Return Values

Sets or returns a **Long** value that indicates the maximum number of records to return. Default is zero (no limit).

Remarks

Use the **MaxRecords** property to limit the number of records that the [provider](#) returns from the data source. The default setting of this property is zero, which means the provider returns all requested records.

The **MaxRecords** property is read/write when the **Recordset** is closed and read-only when it is open.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Mode Property

Indicates the available permissions for modifying data in a [Connection](#), [Record](#), or [Stream](#) object.

Settings and Return Values

Sets or returns a [ConnectModeEnum](#) value. The default value for a **Connection** is **adModeUnknown**. The default value for a **Record** object is **adModeRead**. The default value for a **Stream** associated with an underlying source (opened with a URL as the source, or as the default **Stream** of a **Record**) is **adModeRead**. The default value for a **Stream** not associated with an underlying source (instantiated in memory) is **adModeUnknown**.

Remarks

Use the **Mode** property to set or return the access permissions in use by the [provider](#) on the current connection. You can set the **Mode** property only when the **Connection** object is closed.

For a **Stream** object, if the access mode is not specified, it is inherited from the source used to open the **Stream** object. For example, if a **Stream** is opened from a **Record** object, by default it is opened in the same mode as the **Record**.

This property is read/write while the object is closed and read-only while the object is open.

Remote Data Service Usage When used on a [client-side Connection](#) object, the **Mode** property can only be set to **adModeUnknown**.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Connection Object](#) | [Record Object](#) | [Stream Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Name Property

Indicates the name of an object.

Settings and Return Values

Sets or returns a **String** value that indicates the name of an object.

Remarks

Use the **Name** property to assign a name to or retrieve the name of a **Command**, **Property**, **Field**, or **Parameter** object.

The value is read/write on a **Command** object and read-only on a **Property** object.

For a **Field** object, **Name** is normally read-only. However, for new **Field** objects that have been appended to the [Fields](#) collection of a [Record](#), **Name** is read/write only after the [Value](#) property for the **Field** has been specified and the data provider has successfully added the new **Field** by calling the [Update](#) method of the **Fields** collection.

For **Parameter** objects not yet appended to the [Parameters](#) collection, the **Name** property is read/write. For appended **Parameter** objects and all other objects, the **Name** property is read-only. Names do not have to be unique within a collection.

You can retrieve the **Name** property of an object by an ordinal reference, after which you can refer to the object directly by name. For example, if `rstMain.Properties(20).Name` yields `Updatability`, you can subsequently refer to this property as `rstMain.Properties("Updatability")`.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Command Object](#) | [Field Object](#) | [Parameter Object](#) | [Property Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

NativeError Property

Indicates the [provider](#)-specific error code for a given [Error](#) object.

Return Value

Returns a **Long** value that indicates the error code.

Remarks

Use the **NativeError** property to retrieve the database-specific error information for a particular **Error** object. For example, when using the Microsoft ODBC Provider for OLE DB with a Microsoft SQL Server database, native error codes that originate from SQL Server pass through ODBC and the ODBC Provider to the ADO **NativeError** property.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Error Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Number Property

Indicates the number that uniquely identifies an [Error](#) object.

Return Value

Returns a **Long** value that may correspond to one of the [ErrorValueEnum](#) constants.

Remarks

Use the **Number** property to determine which error occurred. The value of the property is a unique number that corresponds to the error condition.

The [Errors](#) collection returns an HRESULT in either hexadecimal format (for example, 0x80004005) or long value (for example, 2147467259). These HRESULTs can be raised by underlying components, such as OLE DB or even OLE itself. For more information about these numbers, see Chapter 16 of the *OLE DB Programmer's Reference*.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Description Property](#) | [HelpContext, HelpFile Properties](#) | [Source Property \(ADO Error\)](#)

Applies To: [Error Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

NumericScale Property

Indicates the scale of numeric values in a [Parameter](#) or [Field](#) object.

Settings and Return Values

Sets or returns a **Byte** value that indicates the number of decimal places to which numeric values will be resolved.

Remarks

Use the **NumericScale** property to determine how many digits to the right of the decimal point will be used to represent values for a numeric **Parameter** or **Field** object.

For **Parameter** objects, the **NumericScale** property is read/write.

For a **Field** object, **NumericScale** is normally read-only. However, for new **Field** objects that have been appended to the [Fields](#) collection of a [Record](#), **NumericScale** is read/write only after the [Value](#) property for the **Field** has been specified and the data provider has successfully added the new **Field** by calling the [Update](#) method of the **Fields** collection.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Precision Property](#)

Applies To: [Field Object](#) | [Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

OriginalValue Property

Indicates the value of a [Field](#) that existed in the record before any changes were made.

Return Value

Returns a **VARIANT** value that represents the value of a field prior to any change.

Remarks

Use the **OriginalValue** property to return the original field value for a field from the current record.

In *immediate update mode* (in which the [provider](#) writes changes to the underlying data source after you call the [Update](#) method), the **OriginalValue** property returns the field value that existed prior to any changes (that is, since the last **Update** method call). This is the same value that the [CancelUpdate](#) method uses to replace the [Value](#) property.

In *batch update mode* (in which the provider caches multiple changes and writes them to the underlying data source only when you call the [UpdateBatch](#) method), the **OriginalValue** property returns the field value that existed prior to any changes (that is, since the last **UpdateBatch** method call). This is the same value that the [CancelBatch](#) method uses to replace the **Value** property. When you use this property with the [UnderlyingValue](#) property, you can resolve conflicts that arise from batch updates.

Record

For [Record](#) objects, the **OriginalValue** property will be empty for fields added before [Update](#) is called.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[UnderlyingValue Property](#)

Applies To: [Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

PageCount Property

Indicates how many pages of data the [Recordset](#) object contains.

Return Value

Returns a **Long** value that indicates the number of pages in the **Recordset**.

Remarks

Use the **PageCount** property to determine how many pages of data are in the **Recordset** object. *Pages* are groups of records whose size equals the [PageSize](#) property setting. Even if the last page is incomplete because there are fewer records than the **PageSize** value, it counts as an additional page in the **PageCount** value. If the **Recordset** object does not support this property, the value will be -1 to indicate that the **PageCount** is indeterminable.

See the **PageSize** and [AbsolutePage](#) properties for more on page functionality.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AbsolutePage Property](#) | [PageSize Property](#) | [RecordCount Property](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

PageSize Property

Indicates how many records constitute one page in the [Recordset](#).

Settings and Return Values

Sets or returns a **Long** value that indicates how many records are on a page. The default is 10.

Remarks

Use the **PageSize** property to determine how many records make up a logical page of data. Establishing a page size allows you to use the [AbsolutePage](#) property to move to the first record of a particular page. This is useful in [Web-server](#) scenarios when you want to allow the user to page through data, viewing a certain number of records at a time.

This property can be set at any time, and its value will be used for calculating the location of the first record of a particular page.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AbsolutePage Property](#) | [PageCount Property](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ParentRow Property

Sets the container of an OLE DB **Row** object on an **ADOREcordConstruction** object, so that the parent of the row is turned into an ADO **Record** object.

Write-only.

Syntax

```
HRESULT put_ParentRow([in] IUnknown* pParent);
```

Parameters

pParent

A container of a row.

Return Values

This property method returns the standard HRESULT values, including S_OK and E_FAIL.

Applies To

[ADOREcordConstruction](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ParentURL Property

Indicates an [absolute URL](#) string that points to the parent [Record](#) of the current **Record** object.

Return Value

Returns a **String** value that indicates the URL of the parent **Record**.

Remarks

The **ParentURL** property depends upon the source used to open the **Record** object. For example, the **Record** may be opened with a source containing a relative path name of a directory referenced by the [ActiveConnection](#) property.

Suppose "second" is a folder contained under "first". Open the **Record** object with the following:

```
record.ActiveConnection = "http://first"  
record.Open "second"
```

Now, the value of the **ParentURL** property is "http://first", the same as **ActiveConnection**.

The source may also be an absolute URL such as, "http://first/second". The **ParentURL** property is then "http://first", the level above "second".

This property may be a null value if:

- There is no parent for the current object; for example, if the **Record** object represents the root of a directory.
- The **Record** object represents an entity that cannot be specified with a URL.

This property is read-only.

Note This property is only supported by document source providers, such

as the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Records and Provider-Supplied Fields](#).

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

Note If the current record contains a data record from an ADO **Recordset**, accessing the **ParentURL** property causes a run-time error, indicating that no URL is possible.

See Also

Applies To: [Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Position Property

Indicates the current position within a [Stream](#) object.

Settings and Return Values

Sets or returns a **Long** value that specifies the offset, in number of bytes, of the current position from the beginning of the stream. The default is 0, which represents the first byte in the stream.

Remarks

The current position can be moved to a point after the end of the stream. If you specify the current position beyond the end of the stream, the [Size](#) of the **Stream** object will be increased accordingly. Any new bytes added in this way will be null.

Notes **Position** always measures bytes. For text streams using multibyte character sets, multiply the position by the character size to determine the character number. For example, for a two-byte character set, the first character is at position 0, the second character at position 2, the third character at position 4, and so on.

Negative values cannot be used to change the current position in a **Stream**. Only positive numbers can be used for **Position**.

For read-only **Stream** objects, ADO will not return an error if **Position** is set to a value greater than the **Size** of the **Stream**. This does not change the size of the **Stream**, or alter the **Stream** contents in any way. However, doing this should be avoided because it results in a meaningless **Position** value.

See Also

[Example](#)

Charset Property

Applies To: Stream Object

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Precision Property

Indicates the degree of precision for numeric values in a [Parameter](#) object or for numeric [Field](#) objects.

Settings and Return Values

Sets or returns a **Byte** value that indicates the maximum number of digits used to represent values.

Remarks

Use the **Precision** property to determine the maximum number of digits used to represent values for a numeric **Parameter** or **Field** object.

The value is read/write on a **Parameter** object.

For a **Field** object, **Precision** is normally read-only. However, for new **Field** objects that have been appended to the [Fields](#) collection of a [Record](#), **Precision** is read/write only after the [Value](#) property for the **Field** has been specified and the data provider has successfully added the new **Field** by calling the [Update](#) method of the **Fields** collection.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[NumericScale Property](#)

Applies To: [Field Object](#) | [Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Prepared Property

Indicates whether to save a compiled version of a command before execution.

Settings and Return Values

Sets or returns a **Boolean** value that, if set to **True**, indicates that the command should be prepared.

Remarks

Use the **Prepared** property to have the [provider](#) save a prepared (or compiled) version of the query specified in the [CommandText](#) property before a [Command](#) object's first execution. This may slow a command's first execution, but once the provider compiles a command, the provider will use the compiled version of the command for any subsequent executions, which will result in improved performance.

If the property is **False**, the provider will execute the **Command** object directly without creating a compiled version.

If the provider does not support command preparation, it may return an error as soon as this property is set to **True**. If it does not return an error, it simply ignores the request to prepare the command and sets the **Prepared** property to **False**.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Command Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Provider Property

Indicates the name of the [provider](#) for a [Connection](#) object.

Settings and Return Values

Sets or returns a **String** value that indicates the provider name.

Remarks

Use the **Provider** property to set or return the name of the provider for a connection. This property can also be set by the contents of the [ConnectionString](#) property or the *ConnectionString* argument of the [Open](#) method; however, specifying a provider in more than one place while calling the **Open** method can have unpredictable results. If no provider is specified, the property will default to MSDASQL ([Microsoft OLE DB Provider for ODBC](#)).

The **Provider** property is read/write when the connection is closed and read-only when it is open. The setting does not take effect until you either open the **Connection** object or access the [Properties](#) collection of the **Connection** object. If the setting is not valid, an error occurs.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Microsoft OLE DB Provider for ODBC](#) | [Appendix A: Providers](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

RecordCount Property

Indicates the number of records in a [Recordset](#) object.

Return Value

Returns a **Long** value that indicates the number of records in the **Recordset**.

Remarks

Use the **RecordCount** property to find out how many records are in a **Recordset** object. The property returns -1 when ADO cannot determine the number of records or if the provider or [cursor](#) type does not support **RecordCount**. Reading the **RecordCount** property on a closed **Recordset** causes an error.

If the **Recordset** object supports approximate positioning or bookmarks—that is, **Supports (adApproxPosition)** or **Supports (adBookmark)**, respectively, return **True**—this value will be the exact number of records in the **Recordset**, regardless of whether it has been fully populated. If the **Recordset** object does not support approximate positioning, this property may be a significant drain on resources because all records will have to be retrieved and counted to return an accurate **RecordCount** value.

The cursor type of the **Recordset** object affects whether the number of records can be determined. The **RecordCount** property will return -1 for a forward-only cursor; the actual count for a static or keyset cursor; and either -1 or the actual count for a dynamic cursor, depending on the data source.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AbsolutePosition Property](#) | [PageCount Property](#)

Applies To: [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

RecordType Property

Indicates the type of [Record](#) object.

Return Value

Returns a [RecordTypeEnum](#) value.

Remarks

The **RecordType** property is read-only.

See Also

[Example](#)

[Type Property](#) | [Type Property \(ADO Stream\)](#)

Applies To: [Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Row Property

Gets or sets an OLE DB **Row** object from/on an **ADOREcordConstruction** object. When you use **put_Row** to set a **Row** object, a row is turned into an ADO **Record** object.

Read/write.Syntax

```
HRESULT get_Row([out, retval] IUnknown** ppRow);  
HRESULT put_Row([in] IUnknown* pRow);
```

Parameters

ppRow

Pointer to an OLE DB **Row** object.

PRow

An OLE DB **Row** object.

Return Values

This property method returns the standard HRESULT values, including S_OK and E_FAIL.

Applies To

[ADOREcordConstruction](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

RowPosition Property

Gets or sets an OLE DB **RowPosition** object from/on an **ADORecordsetConstruction** object. When you use **put_RowPosition** to set the **RowPosition** object, the resulting **Recordset** object uses the **RowPosition** object to determine the current row.

Read/write.

Syntax

```
HRESULT get_RowPosition([out, retval] IUnknown** ppRowPos);  
HRESULT put_RowPosition([in] IUnknown* pRowPos);
```

Parameters

ppRowPos

Pointer to an OLE DB **RowPosition** object.

pRowPos

An OLE DB **RowPosition** object.

Return Values

This property method returns the standard HRESULT values, including S_OK and E_FAIL.

Remarks

When this property is set, if the **Rowset** object on the **RowPosition** object is different from the **Rowset** object on the **Recordset** object, the former overrides the latter. The same behavior applies to the current **Chapter** of the **RowPosition** as well.

Applies To

[ADORecordsetConstruction](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Rowset Property

Gets or sets an OLE DB **Rowset** object from/on an **ADOREcordsetConstruction** object. When you use `put_Rowset`, the rowset is turned into an ADO **Recordset** object.

Read/write.

Syntax

```
HRESULT get_Rowset([out, retval] IUnknown** ppRowset);  
HRESULT put_Rowset([in] IUnknown* pRowset);
```

Parameters

ppRowset

Pointer to an OLE DB **Rowset** object.

pRowset

An OLE DB **Rowset** object.

Return Values

This property method returns the standard HRESULT values, including S_OK and E_FAIL.

Applies To

[ADOREcordsetConstruction](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Size Property

Indicates the maximum size, in bytes or characters, of a [Parameter](#) object.

Settings and Return Values

Sets or returns a **Long** value that indicates the maximum size in bytes or characters of a value in a **Parameter** object.

Remarks

Use the **Size** property to determine the maximum size for values written to or read from the [Value](#) property of a **Parameter** object.

If you specify a variable-length data type for a **Parameter** object (for example, any **String** type, such as **adVarChar**), you must set the object's **Size** property before appending it to the [Parameters](#) collection; otherwise, an error occurs.

If you have already appended the **Parameter** object to the **Parameters** collection of a [Command](#) object and you change its type to a variable-length data type, you must set the **Parameter** object's **Size** property before executing the **Command** object; otherwise, an error occurs.

If you use the [Refresh](#) method to obtain parameter information from the [provider](#) and it returns one or more variable-length data type **Parameter** objects, ADO may allocate memory for the parameters based on their maximum potential size, which could cause an error during execution. To prevent an error, you should explicitly set the **Size** property for these parameters before executing the command.

The **Size** property is read/write.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#) | [JScriptExample](#)

[Size Property \(ADO Stream\)](#)

Applies To: [Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Size Property (ADO Stream)

Indicates the size of the stream in number of bytes.

Return Values

Returns a **Long** value that specifies the size of the stream in number of bytes. The default value is the size of the stream, or -1 if the size of the stream is not known.

Remarks

Size can be used only with open [Stream](#) objects.

Note Any number of bits can be stored in a **Stream** object, limited only by system resources. If the **Stream** contains more bits than can be represented by a **Long** value, **Size** is truncated and therefore does not accurately represent the length of the **Stream**.

See Also

[Size Property](#)

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Sort Property

Indicates one or more field names on which the [Recordset](#) is sorted, and whether each field is sorted in ascending or descending order.

Settings and Return Values

Sets or returns a **String** value that indicates the field names in the **Recordset** on which to sort. Each name is separated by a comma, and is optionally followed by a blank and the keyword, **ASC**, which sorts the field in ascending order, or **DESC**, which sorts the field in descending order. By default, if no keyword is specified, the field is sorted in ascending order.

Remarks

This property requires the [CursorLocation](#) property to be set to **adUseClient**. A temporary index will be created for each field specified in the **Sort** property if an index does not already exist.

The sort operation is efficient because data is not physically rearranged, but is simply accessed in the order specified by the index.

Setting the **Sort** property to an empty string will reset the rows to their original order and delete temporary indexes. Existing indexes will not be deleted.

Suppose a **Recordset** contains three fields named *firstName*, *middleInitial*, and *lastName*. Set the **Sort** property to the string, "lastName DESC, firstName ASC", which will order the **Recordset** by last name in descending order, then by first name in ascending order. The middle initial is ignored.

No field can be named "ASC" or "DESC" because those names conflict with the keywords **ASC** and **DESC**. Give a field with a conflicting name an alias by using the **AS** keyword in the query that returns the **Recordset**.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Optimize Property—Dynamic \(ADO\)](#) | [SortColumn Property \(RDS\)](#) | [SortDirection Property \(RDS\)](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Source Property (ADO Error)

Indicates the name of the object or application that originally generated an error.

Return Value

Returns a **String** value that indicates the name of an object or application.

Remarks

Use the **Source** property on an [Error](#) object to determine the name of the object or application that originally generated an error. This could be the object's class name or programmatic ID. For errors in ADO, the property value will be **ADODB.ObjectName**, where *ObjectName* is the name of the object that triggered the error. For ADOX and ADO MD, the value will be **ADOX.ObjectName** and **ADOMD.ObjectName**, respectively.

Based on the error documentation from the **Source**, [Number](#), and [Description](#) properties of **Error** objects, you can write code that will handle the error appropriately.

The **Source** property is read-only for **Error** objects.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Description Property](#) | [HelpContext, HelpFile Properties](#) | [Number Property](#) | [Source Property \(ADO Record\)](#) | [Source Property \(ADO Recordset\)](#)

Applies To: [Error Object](#)

ADO 2.5 API Reference

Source Property (ADO Record)

Indicates the data source or object represented by the [Record](#).

Settings and Return Values

Sets or returns a **VARIANT** value that indicates the entity represented by the **Record**.

Remarks

The **Source** property returns the *Source* argument of the **Record** object [Open](#) method. It can contain an [absolute](#) or [relative URL](#) string. An absolute URL can be used without setting the [ActiveConnection](#) property to directly open the **Record** object. An implicit **Connection** object is created in this case.

The **Source** property can also contain a reference to an already open **Recordset**, which opens a **Record** object representing the current row in the **Recordset**.

The **Source** property could also contain a reference to a [Command](#) object which returns a single row of data from the provider.

If the **ActiveConnection** property is also set, then the **Source** property must point to some object that exists within the scope of that connection. For example, in tree-structured namespaces, if the **Source** property contains an absolute URL, it must point to a node that exists inside the scope of the node identified by the URL in the connection string. If the **Source** property contains a relative URL, then it is validated within the context set by the **ActiveConnection** property.

The **Source** property is read/write while the **Record** object is closed, and is read-only while the **Record** object is open.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Source Property \(ADO Error\)](#) | [Source Property \(ADO Recordset\)](#)

Applies To: [Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Source Property (ADO Recordset)

Indicates the data source for a [Recordset](#) object.

Settings and Return Values

Sets a **String** value or [Command](#) object reference; returns only a **String** value that indicates the source of the **Recordset**.

Remarks

Use the **Source** property to specify a data source for a **Recordset** object using one of the following: a **Command** object variable, an SQL statement, a stored procedure, or a table name.

If you set the **Source** property to a **Command** object, the [ActiveConnection](#) property of the **Recordset** object will inherit the value of the **ActiveConnection** property for the specified **Command** object. However, reading the **Source** property does not return a **Command** object; instead, it returns the [CommandText](#) property of the **Command** object to which you set the **Source** property.

If the **Source** property is an SQL statement, a stored procedure, or a table name, you can optimize performance by passing the appropriate *Options* argument with the [Open](#) method call.

The **Source** property is read/write for closed **Recordset** objects and read-only for open **Recordset** objects.

See Also

[Visual Basic Example](#)

[Source Property \(ADO Error\)](#) | [Source Property \(ADO Record\)](#)

Applies To: [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

SQLState Property

Indicates the SQL state for a given [Error](#) object.

Return Value

Returns a five-character **String** value that follows the ANSI SQL standard and indicates the error code.

Remarks

Use the **SQLState** property to read the five-character error code that the [provider](#) returns when an error occurs during the processing of an SQL statement. For example, when using the Microsoft OLE DB Provider for ODBC with a Microsoft SQL Server database, SQL state error codes originate from ODBC, based either on errors specific to ODBC or on errors that originate from Microsoft SQL Server, and are then mapped to ODBC errors. These error codes are documented in the ANSI SQL standard, but may be implemented differently by different data sources.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Error Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

State Property

Indicates for all applicable objects whether the state of the object is open or closed.

Indicates for all applicable objects executing an [asynchronous](#) method, whether the current state of the object is connecting, executing, or retrieving.

Return Value

Returns a **Long** value that can be an [ObjectStateEnum](#) value. The default value is **adStateClosed**.

Remarks

You can use the **State** property to determine the current state of a given object at any time.

The object's **State** property can have a combination of values. For example, if a statement is executing, this property will have a combined value of **adStateOpen** and **adStateExecuting**.

The **State** property is read-only.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Command Object](#) | [Connection Object](#) | [Record Object](#) | [Recordset Object](#) | [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Status Property (ADO Field)

Indicates the status of a [Field](#) object.

Return Value

Returns a [FieldStatusEnum](#) value. The default value is **adFieldOK**.

Remarks

This property always returns **adFieldOK** for fields of a [Recordset](#) object.

Additions and deletions to the [Fields](#) collections of the [Record](#) object are cached until the [Update](#) method is called. The **Status** property enables you to determine which fields have been successfully added or deleted.

To enhance performance, schema changes are cached until **Update** is called, and then the changes are made in a batch optimistic update. If the **Update** method is not called, the server is not updated. If any updates fail then an error is returned and the **Status** property indicates the combined values of the operation and error status code. For example, **adFieldPendingInsert OR adFieldPermissionDenied**. The **Status** property for each **Field** can be used to determine why the **Field** was not added, modified, or deleted. **Status** is only meaningfully exposed on the **Record.Fields** collection and not the **Recordset.Fields** collection.

Two problems can arise when adding, modifying, or deleting a **Field**. If the user deletes a **Field**, it is marked for deletion from the **Fields** collection. If the subsequent **Update** returns an error because the user tried to delete a **Field** for which they do not have permission, the **Field** will have a status of **adFieldPermissionDenied OR adFieldPendingDelete**. Calling the [CancelUpdate](#) method restores original values and sets the **Status** to **adFieldOK**. Similarly, the **Update** method may return an error because a new **Field** was added and given an inappropriate value. In that case the new **Field** will be in the **Fields** collection and have a status of **adFieldPendingInsert** and perhaps **adFieldCantCreate**. You can supply an appropriate value for the new

Field and call **Update** again. Note that calling **Resync** instead requeries the provider.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Field Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Status Property (ADO Recordset)

Indicates the status of the current record with respect to batch updates or other bulk operations.

Return Value

Returns a sum of one or more [RecordStatusEnum](#) values.

Remarks

Use the **Status** property to see what changes are pending for records modified during batch updating. You can also use the **Status** property to view the status of records that fail during bulk operations, such as when you call the [Resync](#), [UpdateBatch](#), or [CancelBatch](#) methods on a [Recordset](#) object, or set the [Filter](#) property on a **Recordset** object to an array of bookmarks. With this property, you can determine how a given record failed and resolve it accordingly.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

StayInSync Property

Indicates, in a hierarchical [Recordset](#) object, whether the reference to the underlying [child](#) records (that is, the *chapter*) changes when the [parent](#) row position changes.

Settings and Return Values

Sets or returns a **Boolean** value. The default value is **True**. If **True**, the chapter will be updated if the parent **Recordset** object changes row position; if **False**, the chapter will continue to refer to data in the previous chapter even though the parent **Recordset** object has changed row position.

Remarks

This property applies to [hierarchical recordsets](#), such as those supported by the [Microsoft Data Shaping Service for OLE DB](#), and must be set on the parent **Recordset** before the child **Recordset** is retrieved. This property simplifies navigating hierarchical recordsets.

See Also

[Visual Basic Example](#)

[Microsoft Data Shaping Service for OLE DB](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Type Property

Indicates the operational type or data type of a [Parameter](#), [Field](#), or [Property](#) object.

Settings and Return Values

Sets or returns a [DataTypeEnum](#) value.

Remarks

For **Parameter** objects, the **Type** property is read/write. For new **Field** objects that have been appended to the [Fields](#) collection of a [Record](#), **Type** is read/write only after the [Value](#) property for the **Field** has been specified and the data provider has successfully added the new **Field** by calling the [Update](#) method of the **Fields** collection.

For all other objects, the **Type** property is read-only.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[RecordType Property](#) | [Type Property \(ADO Stream\)](#)

Applies To: [Field Object](#) | [Parameter Object](#) | [Property Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Type Property (ADO Stream)

Indicates the type of data contained in the [Stream](#) (binary or text).

Settings and Return Values

Sets or returns a [StreamTypeEnum](#) value that specifies the type of data contained in the **Stream** object. The default value is **adTypeText**. However, if binary data is initially written to a new, empty **Stream**, the **Type** will be changed to **adTypeBinary**.

Remarks

The **Type** property is read/write only when the current position is at the beginning of the **Stream** ([Position](#) is 0), and read-only at any other position.

The **Type** property determines which methods should be used for reading and writing the **Stream**. For text **Streams**, use [ReadText](#) and [WriteText](#). For binary **Streams**, use [Read](#) and [Write](#).

See Also

[Example](#)

[RecordType Property](#) | [Type Property](#)

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

UnderlyingValue Property

Indicates a [Field](#) object's current value in the database.

Return Value

Returns a **VARIANT** value that indicates the value of the **Field**.

Remarks

Use the **UnderlyingValue** property to return the current field value from the database. The field value in the **UnderlyingValue** property is the value that is visible to your transaction and may be the result of a recent update by another transaction. This may differ from the [OriginalValue](#) property, which reflects the value that was originally returned to the [Recordset](#).

This is similar to using the [Resync](#) method, but the **UnderlyingValue** property returns only the value for a specific field from the current record. This is the same value that the [Resync](#) method uses to replace the [Value](#) property.

When you use this property with the **OriginalValue** property, you can resolve conflicts that arise from batch updates.

Record

For [Record](#) objects, this property will be empty for fields added before [Update](#) is called.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[OriginalValue Property](#) | [Resync Method](#)

Applies To: [Field Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Value Property

Indicates the value assigned to a [Field](#), [Parameter](#), or [Property](#) object.

Settings and Return Values

Sets or returns a **VARIANT** value that indicates the value of the object. Default value depends on the [Type](#) property.

Remarks

Use the **Value** property to set or return data from **Field** objects, to set or return parameter values with **Parameter** objects, or to set or return property settings with **Property** objects. Whether the **Value** property is read/write or read-only depends upon numerous factors—see the respective object topics for more information.

ADO allows setting and returning long binary data with the **Value** property.

Notes For **Parameter** objects, ADO reads the **Value** property only once from the provider. If a command contains a **Parameter** whose **Value** property is empty, and you create a [Recordset](#) from the command, ensure that you first close the **Recordset** before retrieving the **Value** property. Otherwise, for some [providers](#), the **Value** property may be empty, and won't contain the correct value.

For new **Field** objects that have been appended to the [Fields](#) collection of a [Record](#) object, the **Value** property must be set before any other **Field** properties can be specified. First, a specific value for the **Value** property must have been assigned and [Update](#) on the **Fields** collection called. Then, other properties such as [Type](#) or [Attributes](#) can be accessed.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Field Object](#) | [Parameter Object](#) | [Property Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Version Property

Indicates the ADO version number.

Return Value

Returns a **String** value that indicates the version.

Remarks

Use the **Version** property to return the version number of the ADO implementation.

The version of the provider will be available as a dynamic property in the [Properties](#) collection.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Dynamic Properties

Dynamic properties can be added to the [Properties](#) collections of the [Connection](#), [Command](#), or [Recordset](#) objects. The source for these properties is either a [data provider](#), such as the [OLE DB Provider for SQL Server](#), or a [service provider](#), such as the [Microsoft Cursor Service for OLE DB](#). Refer to the appropriate data provider or service provider documentation for more information about a specific dynamic property.

The [ADO Dynamic Property Index](#) provides a cross-reference between the ADO and OLE DB names for each standard OLE DB provider dynamic property.

The following dynamic properties are of special interest, and are also documented in the sources mentioned above. Special functionality with ADO is documented in the ADO help topics listed below.

[Optimize](#)

Specifies whether an index should be created on this field.

[Prompt](#)

Specifies whether the OLE DB provider should prompt the user for initialization information.

[Reshape Name](#)

Specifies a name for the **Recordset** object.

[Resync Command](#)

Specifies a user-supplied command string that the **Resync** method issues to refresh the data in the table named in the **Unique Table** dynamic property.

Unique Table — specifies the name of the base table upon which updates, insertions, and deletions are allowed.

[Unique Table, Unique Schema, Unique Catalog](#)

Unique Schema — specifies the schema, or name of the owner of the table.

Unique Catalog — specifies the

catalog, or name of the database containing the table.

Specifies whether the **UpdateBatch** method is followed by an implicit **Resync** method operation, and if so, the scope of that operation.

[Update Resync](#)

See Also

[ADO API Reference](#) | [ADO Collections](#) | [ADO Enumerated Constants](#) | [ADO Errors](#) | [ADO Events](#) | [ADO Methods](#) | [ADO Object Model](#) | [ADO Objects](#) | [ADO Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Dynamic Property Index

Data providers, service providers, and service components can add dynamic properties to the **Properties** collections of the unopened [Connection](#) and [Recordset](#) objects. A given provider may also insert additional properties when these objects are opened. Some of these properties are listed in the [ADO Dynamic Properties](#) section. More are listed under the specific providers in the [Appendix A: Providers](#) section.

The table below is a cross-index of the ADO and OLE DB names for each standard OLE DB provider dynamic property. Your providers may add more properties than listed here. For the specific information about provider-specific dynamic properties, see your provider documentation.

The OLE DB Programmer's Reference refers to an ADO property name by the term, "Description." You can find more information about these standard properties in the OLE DB Programmer's Reference. Search for the OLE DB property name in the Index or see the following topics:

- [Appendix C: OLE DB Properties](#)
- [Supported Properties of the Cursor Service](#)
- [Supported Properties of the Persistence Provider](#)
- [Supported OLE DB Properties of the Remoting Provider](#)

Remarks

Note numbers used in the cross-index:

(1) This property is a Boolean flag indicating whether the named interface should be used. The equivalent OLE DB property name is listed if it exists.

(2) The "Bookmarkable" ADO property is generated internally for backwards compatibility, and is mapped to the OLE DB property, DBPROP_IROWSETLOCATE. This is the same property that corresponds to the ADO property, IRowsetLocate.

(3) The ADO property name, "Hidden Columns", is named differently than the

OLE DB property name Description, "Hidden Columns Count."

(4) For hierarchical recordsets, the "Maximum Rows" ADO property gets applied across all children. Depending on the order in which the rows are returned, you might have all, some or no children for each parent or orphaned children in the result set. Therefore, when reshaping hierarchical recordsets, the identifier for every child should be unique. In general, the [Microsoft Data Shaping Service for OLE DB \(MSDATASHAPE\)](#) provider does not allow for distinction between properties that can be inherited from the parent and those that cannot be inherited.

(5) Does not apply.

Connection Dynamic Properties

ADO Property Name	OLE DB Property Name
Active Sessions	DBPROP_ACTIVESESSIONS
Asynchable Abort	DBPROP_ASYNCTXNABORT
Asynchable Commit	DBPROP_ASYNCTNXCOMMIT
Autocommit Isolation Levels	DBPROP_SESS_AUTOCOMMITISOLEVELS
Catalog Location	DBPROP_CATALOGLOCATION
Catalog Term	DBPROP_CATALOGTERM
Column Definition	DBPROP_COLUMNDEFINITION
Connect Timeout	DBPROP_INIT_TIMEOUT
Current Catalog	DBPROP_CURRENTCATALOG
Data Source	DBPROP_INIT_DATASOURCE
Data Source Name	DBPROP_DATASOURCENAME
Data Source Object Threading Model	DBPROP_DSOTHREADMODEL
DBMS Name	DBPROP_DBMSNAME
DBMS Version	DBPROP_DBMSVER
Extended Properties	DBPROP_INIT_PROVIDERSTRING
GROUP BY Support	DBPROP_GROUPBY
Heterogeneous Table Support	DBPROP_HETEROGENEOUSTABLES

Identifier Case Sensitivity	DBPROP_IDENTIFIERCASE
Initial Catalog	DBPROP_INIT_CATALOG
Isolation Levels	DBPROP_SUPPORTEDTXNISOLEVELS
Isolation Retention	DBPROP_SUPPORTEDTXNISORETAIN
Locale Identifier	DBPROP_INIT_LCID
Location	DBPROP_INIT_LOCATION
Maximum Index Size	DBPROP_MAXINDEXSIZE
Maximum Row Size	DBPROP_MAXROWSIZE
Maximum Row Size Includes BLOB	DBPROP_MAXROWSIZEINCLUDESBLOB
Maximum Tables in SELECT	DBPROP_MAXTABLESINSELECT
Mode	DBPROP_INIT_MODE
Multiple Parameter Sets	DBPROP_MULTIPLEPARAMSETS
Multiple Results	DBPROP_MULTIPLERESULTS
Multiple Storage Objects	DBPROP_MULTIPLESTORAGEOBJECTS
Multi-Table Update	DBPROP_MULTITABLEUPDATE
NULL Collation Order	DBPROP_NULLCOLLATION
NULL Concatenation Behavior	DBPROP_CONCATNULLBEHAVIOR
OLE DB Services	DBPROP_INIT_OLEDBSERVICES
OLE DB Version	DBPROP_PROVIDEROLEDBVER
OLE Object Support	DBPROP_OLEOBJECTS
Open Rowset Support	DBPROP_OPENROWSET SUPPORT
ORDER BY Columns in Select List	DBPROP_ORDERBYCOLUMNSINSELECT
Output Parameter Availability	DBPROP_OUTPUTPARAMETERAVAILABILITY
Pass By Ref Accessors	DBPROP_BYREFACCESSORS
Password	DBPROP_AUTH_PASSWORD

Persist Security Info	DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO
Persistent ID Type	DBPROP_PERSISTENTIDTYPE
Prepare Abort Behavior	DBPROP_PREPAREABORTBEHAVIOR
Prepare Commit Behavior	DBPROP_PREPARECOMMITBEHAVIOR
Procedure Term	DBPROP_PROCEDURETERM
Prompt	DBPROP_INIT_PROMPT
Provider Friendly Name	DBPROP_PROVIDERFRIENDLYNAME
Provider Name	DBPROP_PROVIDERFILENAME
Provider Version	DBPROP_PROVIDERVER
Read-Only Data Source	DBPROP_DATASOURCEREADONLY
Rowset Conversions on Command	DBPROP_ROWSETCONVERSIONSONCOMMAND
Schema Term	DBPROP_SCHEMATERM
Schema Usage	DBPROP_SCHEMAUSAGE
SQL Support	DBPROP_SQLSUPPORT
Structured Storage	DBPROP_STRUCTUREDSTORAGE
Subquery Support	DBPROP_SUBQUERIES
Table Term	DBPROP_TABLETERM
Transaction DDL	DBPROP_SUPPORTEDTXNDDL
User ID	DBPROP_AUTH_USERID
User Name	DBPROP_USERNAME
Window Handle	DBPROP_INIT_HWND

Recordset Dynamic Properties

Note that the **Dynamic Properties** of the **Recordset** object go out of scope (become unavailable) when the **Recordset** is closed.

ADO Property Name	OLE DB Property Name
IAccessor	DBPROP_IACCESSOR (1)
IChapteredRowset	(1)

IColumnsInfo	DBPROP_ICOLUMNSINFO (1)
IColumnsRowset	DBPROP_ICOLUMNSROWSET (1)
IConnectionPointContainer	DBPROP_ICONNECTIONPOINTCONTAINER (1)
IConvertType	(1)
ILockBytes	DBPROP_ILOCKBYTES (1)
IRowset	DBPROP_IROWSET (1)
IDBAsynchStatus	DBPROP_IDBASYNCHSTATUS (1)
IParentRowset	(1)
IRowsetChange	DBPROP_IROWSETCHANGE (1)
IRowsetExactScroll	(1)
IRowsetFind	DBPROP_IROWSETFIND (1)
IRowsetIdentity	DBPROP_IROWSETIDENTITY (1)
IRowsetInfo	DBPROP_IROWSETINFO (1)
IRowsetLocate	DBPROP_IROWSETLOCATE (1)
IRowsetRefresh	DBPROP_IROWSETREFRESH (1)
IRowsetResynch	(1)
IRowsetScroll	DBPROP_IROWSETSCROLL (1)
IRowsetUpdate	DBPROP_IROWSETUPDATE (1)
IRowsetView	DBPROP_IROWSETVIEW (1)
IRowsetIndex	DBPROP_IROWSETINDEX (1)
ISequentialStream	DBPROP_ISEQUENTIALSTREAM (1)
IStorage	DBPROP_ISTORAGE (1)
IStream	DBPROP_ISTREAM (1)
ISupportErrorInfo	DBPROP_ISUPPORTERRORINFO (1)
Access Order	DBPROP_ACCESSORDER
Append-Only Rowset	DBPROP_APPENDONLY
Asynchronous Rowset Processing	DBPROP_ROWSET_ASYNCH
Auto Recalc	DBPROP_ADC_AUTORECALC
Background Fetch Size	DBPROP_ASYNCHFETCHSIZE
Background Thread Priority	DBPROP_ASYNCHTHREADPRIORITY
Batch Size	DBPROP_ADC_BATCHSIZE

Blocking Storage Objects	DBPROP_BLOCKINGSTORAGEOBJECTS
Bookmark Type	DBPROP_BOOKMARKTYPE
Bookmarkable	DBPROP_IROWSETLOCATE (2)
Bookmarks Ordered	DBPROP_ORDEREDBOOKMARKS
Cache Child Rows	DBPROP_ADC_CACHECHILDROWS
Cache Deferred Columns	DBPROP_CACHEDEFERRED
Change Inserted Rows	DBPROP_CHANGEINSERTEDROWS
Column Privileges	DBPROP_COLUMNRESTRICT
Column Set Notification	DBPROP_NOTIFYCOLUMNSET
Column Writable	DBPROP_MAYWRITECOLUMN
Command Time Out	DBPROP_COMMANDTIMEOUT
Cursor Engine Version	DBPROP_ADC_CEVER
Defer Column	DBPROP_DEFERRED
Delay Storage Object Updates	DBPROP_DELAYSTORAGEOBJECTS
Fetch Backwards	DBPROP_CANFETCHBACKWARDS
Filter Operations	DBPROP_FILTERCOMPAREOPS
Find Operations	DBPROP_FINDCOMPAREOPS
Hidden Columns (Count)	DBPROP_HIDDENCOLUMNS (3)
Hold Rows	DBPROP_CANHOLDROWS
Immobile Rows	DBPROP_IMMOBILEROWS
Initial Fetch Size	DBPROP_ASYNCHPREFETCHSIZE
Literal Bookmarks	DBPROP_LITERALBOOKMARKS
Literal Row Identity	DBPROP_LITERALIDENTITY
Maintain Change Status	DBPROP_ADC_MAINTAINCHANGESTATUS
Maximum Open Rows	DBPROP_MAXOPENROWS
Maximum Pending Rows	DBPROP_MAXPENDINGROWS
Maximum Rows	DBPROP_MAXROWS (4)
Memory Usage	DBPROP_MEMORYUSAGE
Notification Granularity	DBPROP_NOTIFICATIONGRANULARITY
Notification Phases	DBPROP_NOTIFICATIONPHASES
Objects Transacted	DBPROP_TRANSACTEDOBJECT
Others' Changes Visible	DBPROP_OTHERUPDATEDELETE

Others' Inserts Visible	DBPROP_OTHERINSERT
Own Changes Visible	DBPROP_OWNUPDATEDELETE
Own Inserts Visible	DBPROP_OWNINSERT
Preserve on Abort	DBPROP_ABORTPRESERVE
Preserve on Commit	DBPROP_COMMITPRESERVE
Private1	(5)
Quick Restart	DBPROP_QUICKRESTART
Reentrant Events	DBPROP_REENTRANTEVENTS
Remove Deleted Rows	DBPROP_REMOVEDELETED
Report Multiple Changes	DBPROP_REPORTMULTIPLECHANGES
Reshape Name	DBPROP_ADC_RESHAPENAME
Resync Command	DBPROP_ADC_CUSTOMRESYNCH
Return Pending Inserts	DBPROP_RETURNPENDINGINSERTS
Row Delete Notification	DBPROP_NOTIFYROWDELETE
Row First Change Notification	DBPROP_NOTIFYROWFIRSTCHANGE
Row Insert Notification	DBPROP_NOTIFYROWINSERT
Row Privileges	DBPROP_ROWRESTRICT
Row Resynchronization Notification	DBPROP_NOTIFYROWRESYNCH
Row Threading Model	DBPROP_ROWTHREADMODEL
Row Undo Change Notification	DBPROP_NOTIFYROWUNDOCHANGE
Row Undo Delete Notification	DBPROP_NOTIFYROWUNDODELETE
Row Undo Insert Notification	DBPROP_NOTIFYROWUNDOINSERT
Row Update Notification	DBPROP_NOTIFYROWUPDATE
Rowset Fetch Position Change Notification	DBPROP_NOTIFYROWSETFETCHPOSITIONCH
Rowset Release Notification	DBPROP_NOTIFYROWSETRELEASE
Scroll Backwards	DBPROP_CANSROLLBACKWARDS
Server Cursor	DBPROP_SERVERCURSOR

Skip Deleted Bookmarks	DBPROP_BOOKMARKSKIPPED
Strong Row Identity	DBPROP_STRONGIDENTITY
Unique Catalog	DBPROP_ADC_UNIQUECATALOG
Unique Rows	DBPROP_UNIQUEROWS
Unique Schema	DBPROP_ADC_UNIQUESCHEMA
Unique Table	DBPROP_ADC_UNIQUETABLE
Updatability	DBPROP_UPDATABILITY
Update Criteria	DBPROP_ADC_UPDATECRITERIA
Update Resync	DBPROP_ADC_UPDATERESYNC
Use Bookmarks	DBPROP_BOOKMARKS

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Optimize Property—Dynamic (ADO)

Specifies whether an index should be created on a field.

Settings and Return Values

Sets or returns a **Boolean** value that indicates whether an index should be created.

Remarks

An index can improve the performance of operations that find or sort values in a [Recordset](#). The index is internal to ADO—you cannot explicitly access or use it in your application.

To create an index on a field, set the **Optimize** property to **True**. To delete the index, set this property to **False**.

Optimize is a dynamic property appended to the [Field](#) object [Properties](#) collection when the [CursorLocation](#) property is set to **adUseClient**.

Usage

```
Dim rs As New Recordset
Dim fld As Field
rs.CursorLocation = adUseClient      'Enable index creation
rs.Fields.Append "Field1", adChar, 35, adFldIsNullable
rs.Open
Set fld = rs.Fields(0)
fld.Properties("Optimize") = True    'Create an index
fld.Properties("Optimize") = False   'Delete an index
```

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Filter Property](#) | [Find Method](#) | [Sort Property](#)

Applies To: [Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Prompt Property—Dynamic (ADO)

Specifies whether the OLE DB provider should prompt the user for initialization information.

Settings and Return Values

Sets and returns a [ConnectPromptEnum](#) value.

Remarks

Prompt is a dynamic property, which may be appended to the [Connection](#) object's [Properties](#) collection by the OLE DB provider. To prompt for initialization information, OLE DB providers will typically display a dialog box to the user.

Dynamic properties of a [Connection](#) object are lost when the **Connection** is closed. The **Prompt** property must be reset before re-opening the **Connection** to use a value other than the default.

Note Do 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 seem to lock up.

Usage

```
Set cn = New Connection
cn.Provider = "SQLOLEDB"
cn.Properties("Prompt") = adPromptNever ' do not prompt the user
```

See Also

Applies To: [Connection Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Reshape Name Property—Dynamic (ADO)

Specifies a name for the [Recordset](#) object.

Return Values

Returns a **String** value that is the name of the **Recordset**.

Remarks

Names persist for the duration of the connection or until the **Recordset** is closed.

The **Reshape Name** property is primarily intended for use with the re-shaping feature of the [Microsoft Data Shaping Service for OLE DB service provider](#).

Names must be unique in order to participate in re-shaping.

This property is read-only, but can be set indirectly when a Recordset is created. For example, if a clause of a SHAPE command creates a Recordset and gives it an alias name with the "AS" keyword, then the alias is assigned to the Reshape Name property. If no alias is declared then the reshape name property is assigned a unique name generated by the data shaping service. If the alias name is the same as the name of an existing **Recordset**, neither **Recordset** be reshaped until one of them is released. The default behavior can be changed by setting the "Unique Reshape Names" (See "Microsoft Data shaping service for OLE DB") property on the ADO connection to TRUE. This give the data shaping service permission to change user assignend names if necessary to insure uniqueness.

Use the **Reshape Name** property when you want to refer to a **Recordset** in a SHAPE command, or when you don't know its name because it was generated by Data Shaping Service. In that case, you could generate a SHAPE command by concatenating the command around the string returned by the **Reshape Name** property.

Reshape Name is a dynamic property appended to the **Recordset** object's

[Properties](#) collection when the [CursorLocation](#) property is set to **adUseClient**.

See Also

[Microsoft Data Shaping Service for OLE DB \(Unique Reshape Names\)](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Resync Command Property— Dynamic (ADO)

Specifies a user-supplied command string that the [Resync](#) method issues to refresh the data in the table named in the [Unique Table dynamic property](#).

Settings and Return Values

Sets or returns a **String** value which is a command string.

Remarks

The [Recordset](#) object is the result of a JOIN operation executed on multiple base tables. The rows affected depend on the *AffectRecords* parameter of the [Resync](#) method. The standard **Resync** method is executed if the [Unique Table](#) and **Resync Command** properties are not set.

The command string of the **Resync Command** property is a parameterized command or stored procedure that uniquely identifies the row being refreshed, and returns a single row containing the same number and order of columns as the row to be refreshed. The command string contains a parameter for each primary key column in the **Unique Table**; otherwise, a run-time error is returned. The parameters are automatically filled in with primary key values from the row to be refreshed.

Here are two examples based on SQL:

1) The **Recordset** is defined by a command:

```
SELECT * FROM Customers JOIN Orders ON
    Customers.CustomerID = Orders.CustomerID
WHERE city = Seattle
ORDER BY CustomerID
```

The **Resync Command** property is set to:

```
"SELECT * FROM
  (SELECT * FROM Customers JOIN Orders
   ON Customers.CustomerID = Orders.CustomerID
   city = Seattle ORDER BY CustomerID)
WHERE Orders.OrderID = ?"
```

The **Unique Table** is *Orders* and its primary key, *OrderID*, is parameterized. The sub-select provides a simple way to programmatically ensure that the same number and order of columns are returned as by the original command.

2) The **Recordset** is defined by a stored procedure:

```
CREATE PROC Custorders @CustomerID char(5) AS
SELECT * FROM Customers JOIN Orders ON
Customers.CustomerID = Orders.CustomerID
WHERE Customers.CustomerID = @CustomerID
```

The **Resync** method should execute the following stored procedure:

```
CREATE PROC CustordersResync @ordid int AS
SELECT * FROM Customers JOIN Orders ON
Customers.CustomerID = Orders.CustomerID
WHERE Orders.ordid = @ordid
```

The **Resync Command** property is set to:

```
"{call CustordersResync (?)}"
```

Once again, the **Unique Table** is *Orders* and its primary key, *OrderID*, is parameterized.

Resync Command is a dynamic property appended to the **Recordset** object [Properties](#) collection when the [CursorLocation](#) property is set to **adUseClient**.

See Also

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Unique Table, Unique Schema, Unique Catalog Properties—Dynamic (ADO)

Enables you to closely control modifications to a particular base table in a [Recordset](#) that was formed by a JOIN operation on multiple base tables.

- **Unique Table** specifies the name of the base table upon which updates, insertions, and deletions are allowed.
- **Unique Schema** specifies the *schema*, or name of the owner of the table.
- **Unique Catalog** specifies the *catalog*, or name of the database containing the table.

Settings and Return Values

Sets or returns a **String** value that is the name of a table, schema, or catalog.

Remarks

The desired base table is uniquely identified by its catalog, schema, and table names. When the **Unique Table** property is set, the values of the **Unique Schema** or **Unique Catalog** properties are used to find the base table. It is intended, but not required, that either or both the **Unique Schema** and **Unique Catalog** properties be set before the **Unique Table** property is set.

The primary key of the **Unique Table** is treated as the primary key of the entire **Recordset**. This is the key that is used for any method requiring a primary key.

While **Unique Table** is set, the [Delete](#) method affects only the named table. The [AddNew](#), [Resync](#), [Update](#), and [UpdateBatch](#) methods affect any appropriate underlying base tables of the **Recordset**.

Unique Table must be specified before doing any custom resynchronizations. If **Unique Table** has not been specified, the [Resync Command](#) property will have

no effect.

A run-time error results if a unique base table cannot be found.

These [dynamic properties](#) are all appended to the **Recordset** object [Properties](#) collection when the [CursorLocation](#) property is set to **adUseClient**.

See Also

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Update Resync Property—Dynamic (ADO)

Specifies whether the [UpdateBatch](#) method is followed by an implicit [Resync](#) method operation, and if so, the scope of that operation.

Settings and Return Values

Sets or returns one or more of the [ADCPROP_UPDATERESYNC_ENUM](#) values.

Remarks

The values of `ADCPROP_UPDATERESYNC_ENUM` may be combined, except for `adResyncAll` which already represents the combination of the rest of the values.

The constant `adResyncConflicts` stores the resync values as underlying values, but does not override pending changes.

Update Resync is a dynamic property appended to the [Recordset](#) object [Properties](#) collection when the [CursorLocation](#) property is set to `adUseClient`.

See Also

Applies To: [Recordset Object](#)

ADO 2.5 API Reference

ADO Methods

AddNew	Creates a new record for an updatable Recordset object.
Append	Appends an object to a collection. If the collection is Fields , a new Field object may be created before it is appended to the collection.
AppendChunk	Appends data to a large text or binary data Field , or to a Parameter object.
	Manages transaction processing within a Connection object as follows:
BeginTrans, CommitTrans, and RollbackTrans	BeginTrans — Begins a new transaction. CommitTrans — Saves any changes and ends the current transaction. It may also start a new transaction. RollbackTrans — Cancels any changes and ends the current transaction. It may also start a new transaction.
Cancel	Cancels execution of a pending, asynchronous method call.
CancelBatch	Cancels a pending batch update.
CancelUpdate	Cancels any changes made to the current or new row of a Recordset object, or the Fields collection of a Record object, before calling the Update method.
Clear	Removes all the Error objects from the Errors collection.
Clone	Creates a duplicate Recordset object from an existing Recordset object. Optionally, specifies that the clone be read-only.
Close	Closes an open object and any dependent objects.
CompareBookmarks	Compares two bookmarks and returns an indication of their relative values.
CopyRecord	Copies a file or directory, and its contents, to another location.

CopyTo	Copies the specified number of characters or bytes (depending on Type) in the Stream to another Stream object.
CreateParameter	Creates a new Parameter object with the specified properties.
Delete (ADO Parameters Collection)	Deletes an object from the Parameters collection.
Delete (ADO Fields Collection)	Deletes an object from the Fields collection.
Delete (ADO Recordset)	Deletes the current record or a group of records.
DeleteRecord	Deletes a file or directory, and all its subdirectories.
Execute (ADO Command)	Executes the query, SQL statement, or stored procedure specified in the CommandText property.
Execute (ADO Connection)	Executes the specified query, SQL statement, stored procedure, or provider-specific text.
Find	Searches a Recordset for the row that satisfies the specified criteria.
Flush	Forces the contents of the Stream remaining in the ADO buffer to the underlying object with which the Stream is associated.
GetChildren	Returns a Recordset whose rows represent the files and subdirectories in the directory represented by this Record .
GetChunk	Returns all, or a portion of, the contents of a large text or binary data Field object.
GetRows	Retrieves multiple records of a Recordset object into an array.
GetString	Returns the Recordset as a string.
LoadFromFile	Loads the contents of an existing file into a Stream .
Move	Moves the position of the current record in a Recordset object.
MoveFirst, MoveLast, MoveNext, and MovePrevious	Moves to the first, last, next, or previous record in a specified Recordset object and makes that record the current record.
	Moves a file, or a directory and its contents, to another

<u>MoveRecord</u>	location.
<u>NextRecordset</u>	Clears the current Recordset object and returns the next Recordset by advancing through a series of commands.
<u>Open (ADO Connection)</u>	Opens a connection to a data source.
<u>Open (ADO Record)</u>	Opens an existing Record object, or creates a new file or directory.
<u>Open (ADO Recordset)</u>	Opens a cursor.
<u>Open (ADO Stream)</u>	Opens a Stream object to manipulate streams of binary or text data.
<u>OpenSchema</u>	Obtains database schema information from the provider.
<u>Read</u>	Reads a specified number of bytes from a Stream object.
<u>ReadText</u>	Reads a specified number of characters from a text Stream object.
<u>Refresh</u>	Updates the objects in a collection to reflect objects available from, and specific to, the provider.
<u>Requery</u>	Updates the data in a Recordset object by re-executing the query on which the object is based.
<u>Resync</u>	Refreshes the data in the current Recordset object, or Fields collection of a Record object, from the underlying database.
<u>Save</u>	Saves the Recordset in a file or Stream object.
<u>SaveToFile</u>	Saves the binary contents of a Stream to a file.
<u>Seek</u>	Searches the index of a Recordset to quickly locate the row that matches the specified values, and changes the current row position to that row.
<u>SetEOS</u>	Sets the position that is the end of the stream.
<u>SkipLine</u>	Skips one entire line when reading a text stream.
<u>Stat</u>	Gets statistical information about an open stream.
<u>Supports</u>	Determines whether a specified Recordset object supports a particular type of functionality.

Update	Saves any changes you make to the current row of a Recordset object, or the Fields collection of a Record object.
UpdateBatch	Writes all pending batch updates to disk.
Write	Writes binary data to a Stream object.
WriteText	Writes a specified text string to a Stream object.

See Also

[ADO API Reference](#) | [ADO Collections](#) | [ADO Dynamic Properties](#) | [ADO Enumerated Constants](#) | [ADO Errors](#) | [ADO Events](#) | [ADO Object Model](#) | [ADO Objects](#) | [ADO Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

AddNew Method

Creates a new record for an updatable [Recordset](#) object.

Syntax

```
recordset.AddNew FieldList, Values
```

Parameters

recordset

A **Recordset** object.

FieldList

Optional. A single name, or an [array](#) of names or ordinal positions of the fields in the new record.

Values

Optional. A single value, or an [array](#) of values for the fields in the new record. If *Fieldlist* is an array, *Values* must also be an array with the same number of members; otherwise, an error occurs. The order of field names must match the order of field values in each array.

Remarks

Use the **AddNew** method to create and initialize a new record. Use the [Supports](#) method with **adAddNew** (a [CursorOptionEnum](#) value) to verify whether you can add records to the current **Recordset** object.

After you call the **AddNew** method, the new record becomes the current record and remains current after you call the [Update](#) method. Since the new record is appended to the **Recordset**, a call to **MoveNext** following the Update will move past the end of the **Recordset**, making **EOF** True. If the **Recordset** object does not support bookmarks, you may not be able to access the new record once you move to another record. Depending on your cursor type, you may need to call the [Requery](#) method to make the new record accessible.

If you call **AddNew** while editing the current record or while adding a new

record, ADO calls the **Update** method to save any changes and then creates the new record.

The behavior of the **AddNew** method depends on the updating mode of the **Recordset** object and whether you pass the *Fieldlist* and *Values* arguments.

In *immediate update mode* (in which the provider writes changes to the underlying data source once you call the **Update** method), calling the **AddNew** method without arguments sets the [EditMode](#) property to **adEditAdd** (an [EditModeEnum](#) value). The provider caches any field value changes locally. Calling the **Update** method posts the new record to the database and resets the **EditMode** property to **adEditNone** (an **EditModeEnum** value). If you pass the *Fieldlist* and *Values* arguments, ADO immediately posts the new record to the database (no **Update** call is necessary); the **EditMode** property value does not change (**adEditNone**).

In *batch update mode* (in which the provider caches multiple changes and writes them to the underlying data source only when you call the [UpdateBatch](#) method), calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. The provider caches any field value changes locally. Calling the **Update** method adds the new record to the current **Recordset** and resets the **EditMode** property to **adEditNone**, but the provider does not post the changes to the underlying database until you call the **UpdateBatch** method. If you pass the *Fieldlist* and *Values* arguments, ADO sends the new record to the provider for storage in a cache; you need to call the **UpdateBatch** method to post the new record to the underlying database.

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[SafeArray](#) | [CancelUpdate Method](#) | [EditMode Property](#) | [Requery Method](#) | [Supports Method](#) | [Update Method](#) | [UpdateBatch Method](#)

Applies To: [Recordset Object](#)

ADO 2.5 API Reference

Append Method

Appends an object to a collection. If the collection is [Fields](#), a new [Field](#) object may be created before it is appended to the collection.

Syntax

```
collection.Append object  
fields.Append Name, Type, DefinedSize, Attrib, FieldValue
```

Parameters

collection

A collection object.

fields

A **Fields** collection.

object

An object variable that represents the object to be appended.

Name

A **String** value that contains the name of the new **Field** object, and must not be the same name as any other object in *fields*.

Type

A [DataTypeEnum](#) value, whose default value is **adEmpty**, that specifies the data type of the new field. The following data types are not supported by ADO, and should not be used when appending new fields to a **Recordset**: **adIDispatch**, **adIUnknown**, **adVariant**.

DefinedSize

Optional. A **Long** value that represents the defined size, in characters or bytes, of the new field. The default value for this parameter is derived from *Type*. Fields with a *DefinedSize* greater than 255 bytes, and treated as variable length columns. (The default *DefinedSize* is unspecified.)

Attrib

Optional. A [FieldAttributeEnum](#) value, whose default value is **adFldDefault**, that specifies attributes for the new field. If this value is not specified, the field will contain attributes derived from *Type*.

FieldValue

Optional. A **Variant** that represents the value for the new field. If not specified, then the field is appended with a null value.

Remarks

Parameters Collection

You must set the [Type](#) property of a [Parameter](#) object before appending it to the **Parameters** collection. If you select a variable-length data type, you must also set the [Size](#) property to a value greater than zero.

Describing parameters yourself minimizes calls to the provider and consequently improves performance when using stored procedures or [parameterized](#) queries. However, you must know the properties of the parameters associated with the stored procedure or parameterized query that you want to call.

Use the [CreateParameter](#) method to create **Parameter** objects with the appropriate property settings and use the **Append** method to add them to the [Parameters](#) collection. This lets you set and return parameter values without having to call the provider for the parameter information. If you are writing to a provider that does not supply parameter information, you must use this method to manually populate the **Parameters** collection in order to use parameters at all.

Fields Collection

The *FieldValue* parameter is only valid when adding a **Field** object to a [Record](#) object, not to a **Recordset** object. With a **Record** object, you may append fields and provide values at the same time. With a **Recordset** object, you must create fields while the **Recordset** is closed, then open the **Recordset** and assign values to the fields.

Notes For new **Field** objects that have been appended to the **Fields** collection of a **Record** object, the [Value](#) property must be set before any other **Field** properties can be specified. First, a specific value for the **Value** property must have been assigned and [Update](#) on the **Fields** collection called. Then, other properties such as [Type](#) or [Attributes](#) can be accessed.

Field objects of the following data types (**DataTypeEnum**) cannot be appended to the **Fields** collection and will cause an error to occur:

adArray, **adChapter**, **adEmpty**, **adPropVariant**, and **adUserDefined**. Also, the following data types are not supported by ADO: **adIDispatch**, **adIUnknown**, and **adIVariant**. For these types, no error will occur when appended, but usage can produce unpredictable results including memory leaks.

Recordset

If you do not set the [CursorLocation](#) property before calling the **Append** method, **CursorLocation** will be set to **adUseClient** (a [CursorLocationEnum](#) value) automatically when the [Recordset](#) object's [Open](#) method is called.

A run-time error will occur if the **Append** method is called on the **Fields** collection of an open **Recordset**, or on a **Recordset** where the [ActiveConnection](#) property has been set. You can only append fields to a **Recordset** that is not open and has not yet been connected to a data source. This is typically the case when a **Recordset** object is fabricated with the [CreateRecordset](#) method or assigned to an object variable.

Record

A run-time error will not occur if the **Append** method is called on the **Fields** collection of an open **Record**. The new field will be added to the **Record** object's **Fields** collection. If the **Record** was derived from a **Recordset**, then the new field will not appear in the **Recordset** object's **Fields** collection.

A non-existent field can be created and appended to the **Fields** collection by assigning a value to the field object as if it already existed in the collection. The assignment will trigger the automatic creation and appending of the **Field** object, then the assignment will be completed.

After appending a **Field** to a **Record** object's **Fields** collection, call the **Update** method of the **Fields** collection to save the change.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[CreateParameter Method](#) | [Delete Method \(ADO Fields Collection\)](#) | [Delete](#)

[Method \(ADO Parameters Collection\)](#) | [Delete Method \(ADO Recordset\)](#) | [Update Method](#)

Applies To: [Fields Collection](#) | [Parameters Collection](#) | [Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

AppendChunk Method

Appends data to a large text or binary data [Field](#), or to a [Parameter](#) object.

Syntax

object.AppendChunk *Data*

Parameters

object

A **Field** or **Parameter** object.

Data

A **Variant** that contains the data to append to the object.

Remarks

Use the **AppendChunk** method on a **Field** or **Parameter** object to fill it with long binary or character data. In situations where system memory is limited, you can use the **AppendChunk** method to manipulate long values in portions rather than in their entirety.

Field

If the **adFldLong** bit in the [Attributes](#) property of a **Field** object is set to true, you can use the **AppendChunk** method for that field.

The first **AppendChunk** call on a **Field** object writes data to the field, overwriting any existing data. Subsequent **AppendChunk** calls add to existing data. If you are appending data to one field and then you set or read the value of another field in the current record, ADO assumes that you are finished appending data to the first field. If you call the **AppendChunk** method on the first field again, ADO interprets the call as a new **AppendChunk** operation and overwrites the existing data. Accessing fields in other [Recordset](#) objects that are not clones of the first **Recordset** object will not disrupt **AppendChunk** operations.

If there is no current record when you call **AppendChunk** on a **Field** object, an error occurs.

Note The **AppendChunk** method does not operate on **Field** objects of a [Record](#) object. It does not perform any operation and will produce a run-time error.

Parameter

If the **adParamLong** bit in the **Attributes** property of a **Parameter** object is set to true, you can use the **AppendChunk** method for that parameter.

The first **AppendChunk** call on a **Parameter** object writes data to the parameter, overwriting any existing data. Subsequent **AppendChunk** calls on a **Parameter** object add to existing parameter data. An **AppendChunk** call that passes a null value discards all of the parameter data.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Attributes Property](#) | [GetChunk Method](#)

Applies To: [Field Object](#) | [Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

BeginTrans, CommitTrans, and RollbackTrans Methods

These transaction methods manage transaction processing within a [Connection](#) object as follows:

- **BeginTrans** — Begins a new transaction.
- **CommitTrans** — Saves any changes and ends the current transaction. It may also start a new transaction.
- **RollbackTrans** — Cancels any changes made during the current transaction and ends the transaction. It may also start a new transaction.

Syntax

```
level = object.BeginTrans()  
object.BeginTrans  
object.CommitTrans  
object.RollbackTrans
```

Return Value

BeginTrans can be called as a function that returns a **Long** variable indicating the nesting level of the transaction.

Parameters

object

A **Connection** object.

Connection

Use these methods with a **Connection** object when you want to save or cancel a series of changes made to the source data as a single unit. For example, to transfer money between accounts, you subtract an amount from one and add the same amount to the other. If either update fails, the accounts no longer balance.

Making these changes within an open transaction ensures that either all or none of the changes go through.

Note Not all providers support transactions. Verify that the provider-defined property "**Transaction DDL**" appears in the **Connection** object's [Properties](#) collection, indicating that the provider supports transactions. If the provider does not support transactions, calling one of these methods will return an error.

After you call the **BeginTrans** method, the provider will no longer instantaneously commit changes you make until you call **CommitTrans** or **RollbackTrans** to end the transaction.

For providers that support nested transactions, calling the **BeginTrans** method within an open transaction starts a new, nested transaction. The return value indicates the level of nesting: a return value of "1" indicates you have opened a top-level transaction (that is, the transaction is not nested within another transaction), "2" indicates that you have opened a second-level transaction (a transaction nested within a top-level transaction), and so forth. Calling **CommitTrans** or **RollbackTrans** affects only the most recently opened transaction; you must close or roll back the current transaction before you can resolve any higher-level transactions.

Calling the **CommitTrans** method saves changes made within an open transaction on the connection and ends the transaction. Calling the **RollbackTrans** method reverses any changes made within an open transaction and ends the transaction. Calling either method when there is no open transaction generates an error.

Depending on the **Connection** object's [Attributes](#) property, calling either the **CommitTrans** or **RollbackTrans** methods may automatically start a new transaction. If the **Attributes** property is set to **adXactCommitRetaining**, the provider automatically starts a new transaction after a **CommitTrans** call. If the **Attributes** property is set to **adXactAbortRetaining**, the provider automatically starts a new transaction after a **RollbackTrans** call.

Remote Data Service

The **BeginTrans**, **CommitTrans**, and **RollbackTrans** methods are not available on a client-side **Connection** object.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Attributes Property](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Cancel Method

Cancels execution of a pending, [asynchronous](#) method call.

Syntax

object.**Cancel**

Remarks

Use the **Cancel** method to terminate execution of an asynchronous method call (that is, a method invoked with the **adAsyncConnect**, **adAsyncExecute**, or **adAsyncFetch** option).

The following table shows what task is terminated when you use the **Cancel** method on a particular type of object.

If <i>object</i> is a	The last asynchronous call to this method is terminated
Command	Execute
Connection	Execute or Open
Record	CopyRecord , DeleteRecord , MoveRecord , or Open
Recordset	Open
Stream	Open

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Cancel Method \(RDS\)](#) | [CancelBatch Method](#) | [CancelUpdate Method](#) | [CancelUpdate Method \(RDS\)](#) | [Execute Method \(ADO Command\)](#) | [Execute Method \(ADO Connection\)](#) | [Open Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#)

Applies To: [Command Object](#) | [Connection Object](#) | [Record Object](#) | [Recordset Object](#) | [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CancelBatch Method

Cancels a pending batch update.

Syntax

```
recordset.CancelBatch AffectRecords
```

Parameters

AffectRecords

Optional. An [AffectEnum](#) value that indicates how many records the **CancelBatch** method will affect.

Remarks

Use the **CancelBatch** method to cancel any pending updates in a [Recordset](#) in batch update mode. If the **Recordset** is in immediate update mode, calling **CancelBatch** without **adAffectCurrent** generates an error.

If you are editing the current record or are adding a new record when you call **CancelBatch**, ADO first calls the [CancelUpdate](#) method to cancel any cached changes. After that, all pending changes in the **Recordset** are canceled.

It's possible that the current record will be indeterminable after a **CancelBatch** call, especially if you were in the process of adding a new record. For this reason, it is prudent to set the current record position to a known location in the **Recordset** after the **CancelBatch** call. For example, call the [MoveFirst](#) method.

If the attempt to cancel the pending updates fails because of a conflict with the underlying data (for example, a record has been deleted by another user), the provider returns warnings to the [Errors](#) collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. Use the [Filter](#) property (**adFilterAffectedRecords**) and the [Status](#) property to locate records with conflicts.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#)

[Cancel Method](#) | [Cancel Method \(RDS\)](#) | [CancelUpdate Method](#) | [CancelUpdate Method \(RDS\)](#) | [Clear Method](#) | [LockType Property](#) | [UpdateBatch Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CancelUpdate Method

Cancels any changes made to the current or new row of a [Recordset](#) object, or the [Fields](#) collection of a [Record](#) object, before calling the [Update](#) method.

Syntax

```
recordset.CancelUpdate  
record.Fields.CancelUpdate
```

Remarks

Recordset

Use the **CancelUpdate** method to cancel any changes made to the current row or to discard a newly added row. You cannot cancel changes to the current row or a new row after you call the **Update** method, unless the changes are either part of a transaction that you can roll back with the [RollbackTrans](#) method, or part of a batch update. In the case of a batch update, you can cancel the **Update** with the **CancelUpdate** or [CancelBatch](#) method.

If you are adding a new row when you call the **CancelUpdate** method, the current row becomes the row that was current before the [AddNew](#) call.

If you are in edit mode and want to move off the current record (for example, with [Move](#), [NextRecordset](#), or [Close](#)), you can use **CancelUpdate** to cancel any pending changes. You may need to do this if the update cannot successfully be posted to the data source (for example, an attempted delete that fails due to referential integrity violations will leave the **Recordset** in edit mode after a call to [Delete](#)).

Record

The **CancelUpdate** method cancels any pending insertions or deletions of [Field](#) objects, and cancels pending updates of existing fields and restores them to their original values. The [Status](#) property of all fields in the **Fields** collection is set to **adFieldOK**.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AddNew Method](#) | [Cancel Method](#) | [Cancel Method \(RDS\)](#) | [CancelBatch Method](#) | [CancelUpdate Method \(RDS\)](#) | [EditMode Property](#) | [Update Method](#)

Applies To: [Fields Collection](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Clear Method

Removes all the **Error** objects from the **Errors** collection.

Syntax

Errors.Clear

Remarks

Use the **Clear** method on the [Errors](#) collection to remove all existing [Error](#) objects from the collection. When an error occurs, ADO automatically clears the **Errors** collection and fills it with **Error** objects based on the new error.

Some properties and methods return warnings that appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the [Resync](#), [UpdateBatch](#), or [CancelBatch](#) methods on a [Recordset](#) object; the [Open](#) method on a [Connection](#) object; or set the [Filter](#) property on a **Recordset** object, call the **Clear** method on the **Errors** collection. That way, you can read the [Count](#) property of the **Errors** collection to test for returned warnings.

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[CancelBatch Method](#) | [Delete Method \(ADO Fields Collection\)](#) | [Delete Method \(ADO Parameters Collection\)](#) | [Delete Method \(ADO Recordset\)](#) | [Filter Property](#) | [Resync Method](#) | [UpdateBatch Method](#)

Applies To: [Errors Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Clone Method

Creates a duplicate [Recordset](#) object from an existing **Recordset** object. Optionally, specifies that the clone be read-only.

Syntax

```
Set rstDuplicate = rstOriginal.Clone (LockType)
```

Return Value

Returns a **Recordset** object reference.

Parameters

rstDuplicate

An [object variable](#) that identifies the duplicate **Recordset** object to be created.

rstOriginal

An object variable that identifies the **Recordset** object to be duplicated.

LockType

Optional. A [LockTypeEnum](#) value that specifies either the lock type of the original **Recordset**, or a read-only **Recordset**. Valid values are **adLockUnspecified** or **adLockReadOnly**.

Remarks

Use the **Clone** method to create multiple, duplicate **Recordset** objects, particularly if you want to maintain more than one current record in a given set of records. Using the **Clone** method is more efficient than creating and opening a new **Recordset** object with the same definition as the original.

The [Filter](#) property of the original **Recordset**, if any, will not be applied to the clone. Set the **Filter** property of the new **Recordset** in order to filter the results. The simplest way to copy any existing **Filter** value is to assign it directly, like

```
this: rsNew.Filter = rsOriginal.Filter
```

The current record of a newly created clone is set to the first record.

Changes you make to one **Recordset** object are visible in all of its clones regardless of [cursor](#) type. However, after you execute [Requery](#) on the original **Recordset**, the clones will no longer be synchronized to the original.

Closing the original **Recordset** does not close its copies, nor does closing a copy close the original or any of the other copies.

You can only clone a **Recordset** object that supports bookmarks. Bookmark values are interchangeable; that is, a bookmark reference from one **Recordset** object refers to the same record in any of its clones.

Some **Recordset** events that are triggered will also fire in all **Recordset** clones. However, because the current record can differ between cloned **Recordsets**, the events may not be valid for the clone.

For example, if you change a value of a field, a [WillChangeField](#) event will occur in the changed **Recordset** and in all clones. The *Fields* parameter of the **WillChangeField** event of a cloned **Recordset** (where the change was not made) will simply refer to the fields of the current record of the clone, which may be a different record than the current record of the original **Recordset** where the change occurred.

The following table provided a full listing of all **Recordset** events and indicates whether they are valid and triggered for any recordset clones generated using the **Clone** method.

Event	Triggered in clones?
EndOfRecordset	No
FetchComplete	No
FetchProgress	No
FieldChangeComplete	Yes
MoveComplete	No
RecordChangeComplete	Yes
RecordsetChangeComplete	No

WillChangeField	Yes
WillChangeRecord	Yes
WillChangeRecordset	No
WillMove	No

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Close Method

Closes an open object and any dependent objects.

Syntax

object.**Close**

Remarks

Use the **Close** method to close a [Connection](#), a [Record](#), a [Recordset](#), or a [Stream](#) object to free any associated system resources. Closing an object does not remove it from memory; you can change its property settings and open it again later. To completely eliminate an object from memory, set the [object variable](#) to *Nothing* (in Visual Basic) after closing the object.

Connection

Using the **Close** method to close a **Connection** object also closes any active **Recordset** objects associated with the connection. A [Command](#) object associated with the **Connection** object you are closing will persist, but it will no longer be associated with a **Connection** object; that is, its [ActiveConnection](#) property will be set to **Nothing**. Also, the **Command** object's [Parameters](#) collection will be cleared of any provider-defined parameters.

You can later call the [Open](#) method to re-establish the connection to the same, or another, data source. While the **Connection** object is closed, calling any methods that require an open connection to the data source generates an error.

Closing a **Connection** object while there are open **Recordset** objects on the connection rolls back any pending changes in all of the **Recordset** objects. Explicitly closing a **Connection** object (calling the **Close** method) while a transaction is in progress generates an error. If a **Connection** object falls out of scope while a transaction is in progress, ADO automatically rolls back the transaction.

Recordset, Record, Stream

Using the **Close** method to close a **Recordset**, **Record**, or **Stream** object releases the associated data and any exclusive access you may have had to the data through this particular object. You can later call the [Open](#) method to reopen the object with the same, or modified, attributes.

While a **Recordset** object is closed, calling any methods that require a live [cursor](#) generates an error.

If an edit is in progress while in immediate update mode, calling the **Close** method generates an error; instead, call the [Update](#) or [CancelUpdate](#) method first. If you close the **Recordset** object while in batch update mode, all changes since the last [UpdateBatch](#) call are lost.

If you use the [Clone](#) method to create copies of an open **Recordset** object, closing the original or a clone does not affect any of the other copies.

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Open Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#) | [Save Method](#)

Applies To: [Connection Object](#) | [Record Object](#) | [Recordset Object](#) | [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CompareBookmarks Method

Compares two bookmarks and returns an indication of their relative values.

Syntax

```
result = recordset.CompareBookmarks(Bookmark1, Bookmark2)
```

Return Value

Returns a [CompareEnum](#) value that indicates the relative row position of two records represented by their bookmarks.

Parameters

Bookmark1

The bookmark of the first row.

Bookmark2

The bookmark of the second row.

Remarks

The bookmarks must apply to the same [Recordset](#) object, or a **Recordset** object and its [clone](#). You cannot reliably compare bookmarks from different **Recordset** objects, even if they were created from the same source or command. Nor can you compare bookmarks for a **Recordset** object whose underlying provider does not support comparisons.

A bookmark uniquely identifies a row in a **Recordset** object. Use the current row's [Bookmark](#) property to obtain its bookmark.

Because the data type of a bookmark is provider specific, ADO exposes it as a Variant. For example, SQL Server bookmarks are of type DBTYPE_R8 (Double). ADO would expose this type as a Variant with a subtype of Double.

When comparing bookmarks, ADO does not attempt any type of coercion. The

values are simply passed to the provider where the compare occurs. If bookmarks passed to the **CompareBookmarks** method are stored in variables of differing types, it can generate the type mismatch error, "Arguments are of the wrong type, are out of the acceptable range, or are in conflict with each other."

A bookmark that is not valid or incorrectly formed will cause an error.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Bookmark Property](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CopyRecord Method

Copies a entity represented by a **Record** to another location.

Syntax

Record.**CopyRecord** (*Source*, *Destination*, *UserName*, *Password*, *Options*,

Parameters

Source

Optional. A **String** value that contains a URL specifying the entity to be copied (for example, a file or directory). If *Source* is omitted or specifies an empty string, the file or directory represented by the current [Record](#) will be copied.

Destination

Optional. A **String** value that contains a URL specifying the location where *Source* will be copied.

UserName

Optional. A **String** value that contains the user ID that, if needed, authorizes access to *Destination*.

Password

Optional. A **String** value that contains the password that, if needed, verifies *UserName*.

Options

Optional. A [CopyRecordOptionsEnum](#) value that has a default value of **adCopyUnspecified**. Specifies the behavior of this method.

Async

Optional. A **Boolean** value that, when **True**, specifies that this operation should be [asynchronous](#).

Return Value

A **String** value that typically returns the value of *Destination*. However, the exact value returned is provider-dependent.

Remarks

The values of *Source* and *Destination* must not be identical; otherwise, a run-time error occurs. At least one of the server, path, or resource names must differ.

All children (for example, subdirectories) of *Source* are copied recursively, unless **adCopyNonRecursive** is specified. In a recursive operation, *Destination* must not be a subdirectory of *Source*; otherwise, the operation will not complete.

This method fails if *Destination* identifies an existing entity (for example, a file or directory), unless **adCopyOverWrite** is specified.

Important Use the **adCopyOverWrite** option judiciously. For example, specifying this option when copying a file to a directory will *delete* the directory and replace it with the file.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

Applies To: [Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CopyTo Method

Copies the specified number of characters or bytes (depending on [Type](#)) in the [Stream](#) to another **Stream** object.

Syntax

```
Stream.CopyTo DestStream, NumChars
```

Parameters

DestStream

An object variable value that contains a reference to an open **Stream** object. The current **Stream** is copied to the destination **Stream** specified by *DestStream*. The destination **Stream** must already be open. If not, a run-time error occurs.

Note The *DestStream* parameter may not be a proxy of **Stream** object because this requires access to a private interface on the **Stream** object that cannot be remoted to the client.

NumChars

Optional. An **Integer** value that specifies the number of bytes or characters to be copied from the current position in the source **Stream** to the destination **Stream**. The default value is -1 , which specifies that all characters or bytes are copied from the current position to [EOS](#).

Remarks

This method copies the specified number of characters or bytes, starting from the current position specified by the [Position](#) property. If the specified number is more than the available number of bytes until **EOS**, then only characters or bytes from the current position to **EOS** are copied. If the value of *NumChars* is -1 , or omitted, all characters or bytes starting from the current position are copied.

If there are existing characters or bytes in the destination stream, all contents

beyond the point where the copy ends remain, and are not truncated. **Position** becomes the byte immediately following the last byte copied. If you want to truncate these bytes, call [SetEOS](#).

CopyTo should be used to copy data to a destination **Stream** of the same type as the source **Stream** (their **Type** property settings are both **adTypeText** or both **adTypeBinary**). For text **Stream** objects, you can change the [Charset](#) property setting of the destination **Stream** to translate from one character set to another. Also, text **Stream** objects can be successfully copied into binary **Stream** objects, but binary **Stream** objects cannot be copied into text **Stream** objects.

See Also

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CreateParameter Method

Creates a new [Parameter](#) object with the specified properties.

Syntax

```
Set parameter = command.CreateParameter (Name, Type, Direction, Size
```

Return Value

Returns a **Parameter** object.

Parameters

Name

Optional. A **String** value that contains the name of the **Parameter** object.

Type

Optional. A [DataTypeEnum](#) value that specifies the data type of the **Parameter** object.

Direction

Optional. A [ParameterDirectionEnum](#) value that specifies the type of **Parameter** object.

Size

Optional. A **Long** value that specifies the maximum length for the parameter value in characters or bytes.

Value

Optional. A **VARIANT** that specifies the value for the **Parameter** object.

Remarks

Use the **CreateParameter** method to create a new **Parameter** object with a specified name, type, direction, size, and value. Any values you pass in the arguments are written to the corresponding **Parameter** properties.

This method does not automatically append the **Parameter** object to the

Parameters collection of a [Command](#) object. This lets you set additional properties whose values ADO will validate when you append the **Parameter** object to the collection.

If you specify a variable-length data type in the *Type* argument, you must either pass a *Size* argument or set the [Size](#) property of the **Parameter** object before appending it to the **Parameters** collection; otherwise, an error occurs.

If you specify a numeric data type (**adNumeric** or **adDecimal**) in the *Type* argument, then you must also set the [NumericScale](#) and [Precision](#) properties.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Append Method](#) | [Parameter Object](#) | [Parameters Collection](#)

Applies To: [Command Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Delete Method (ADO Parameters Collection)

Deletes an object from the [Parameters](#) collection.

Syntax

Parameters.Delete *Index*

Parameters

Index

A **String** value that contains the name of the object you want to delete, or the objects ordinal position (index) in the collection.

Remarks

Using the **Delete** method on a collection lets you remove one of the objects in the collection. This method is available only on the **Parameters** collection of a [Command](#) object. You must use the [Parameter](#) object's [Name](#) property or its collection index when calling the **Delete** method—an [object variable](#) is not a valid argument.

See Also

[Delete Method \(ADO Fields Collection\)](#) | [Delete Method \(ADO Recordset\)](#) | [DeleteRecord Method](#)

Applies To: [Parameters Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Delete Method (ADO Fields Collection)

Deletes an object from the [Fields](#) collection.

Syntax

```
Fields.Delete Field
```

Parameters

Field

A **Variant** that designates the [Field](#) object to delete. This parameter can be the name of the **Field** object or the ordinal position of the **Field** object itself.

Remarks

Calling the **Fields.Delete** method on an open [Recordset](#) causes a run-time error.

See Also

[Delete Method \(ADO Parameters Collection\)](#) | [Delete Method \(ADO Recordset\)](#)
| [DeleteRecord Method](#)

Applies To: [Fields Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Delete Method (ADO Recordset)

Deletes the current record or a group of records.

Syntax

```
recordset.Delete AffectRecords
```

Parameters

AffectRecords

An [AffectEnum](#) value that determines how many records the **Delete** method will affect. The default value is **adAffectCurrent**.

Note **adAffectAll** and **adAffectAll Chapters** are not valid arguments to **Delete**.

Remarks

Using the **Delete** method marks the current record or a group of records in a [Recordset](#) object for deletion. If the **Recordset** object doesn't allow record deletion, an error occurs. If you are in immediate update mode, deletions occur in the database immediately. If a record cannot be successfully deleted (due to database integrity violations, for example), the record will remain in edit mode after the call to **Update**. This means that you must cancel the update with [CancelUpdate](#) before moving off the current record (for example, with [Close](#), [Move](#), or [NextRecordset](#)).

If you are in batch update mode, the records are marked for deletion from the cache and the actual deletion happens when you call the [UpdateBatch](#) method. (Use the [Filter](#) property to view the deleted records.)

Retrieving field values from the deleted record generates an error. After deleting the current record, the deleted record remains current until you move to a different record. Once you move away from the deleted record, it is no longer accessible.

If you nest deletions in a transaction, you can recover deleted records with the [RollbackTrans](#) method. If you are in batch update mode, you can cancel a pending deletion or group of pending deletions with the [CancelBatch](#) method.

If the attempt to delete records fails because of a conflict with the underlying data (for example, a record has already been deleted by another user), the [provider](#) returns warnings to the [Errors](#) collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records.

If the [Unique Table dynamic property](#) is set, and the **Recordset** is the result of executing a JOIN operation on multiple tables, then the **Delete** method will only delete rows from the table named in the [Unique Table](#) property.

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Delete Method \(ADO Fields Collection\)](#) | [Delete Method \(ADO Parameters Collection\)](#) | [DeleteRecord Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

DeleteRecord Method

Deletes a the entity represented by a [Record](#).

Syntax

```
Record.DeleteRecord Source, Async
```

Parameters

Source

Optional. A **String** value that contains a URL identifying the entity (for example, the file or directory) to be deleted. If *Source* is omitted or specifies an empty string, the entity represented by the current [Record](#) is deleted. If the *Record* is a collection record ([RecordType](#) of **adCollectionRecord**, such as a directory) all children (for example, subdirectories) will also be deleted.

Async

Optional. A **Boolean** value that, when **True**, specifies that the delete operation is [asynchronous](#).

Remarks

Operations on the object represented by this **Record** may fail after this method completes. After calling **DeleteRecord**, the **Record** should be closed because the behavior of the **Record** may become unpredictable depending upon when the provider updates the **Record** with the data source.

If this **Record** was obtained from a [Recordset](#), then the results of this operation will not be reflected immediately in the **Recordset**. Refresh the **Recordset** by closing and re-opening it, or by executing the **Recordset** [Requery](#), or [Update](#) and [Resync](#) methods.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Delete Method \(ADO Fields Collection\)](#) | [Delete Method \(ADO Parameters Collection\)](#) | [Delete Method \(ADO Recordset\)](#)

Applies To: [Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Execute Method (ADO Command)

Executes the query, SQL statement, or stored procedure specified in the [CommandText](#) property.

Syntax

For a **Recordset**-returning **Command**:

```
Set recordset = command.Execute( RecordsAffected, Parameters, Option
```

For a non-recordset-returning **Command**:

```
command.Execute RecordsAffected, Parameters, Options
```

Return Value

Returns a [Recordset](#) object reference or **Nothing**.

Parameters

RecordsAffected

Optional. A **Long** variable to which the [provider](#) returns the number of records that the operation affected. The *RecordsAffected* parameter applies only for action queries or stored procedures. *RecordsAffected* does not return the number of records returned by a result-returning query or stored procedure. To obtain this information, use the [RecordCount](#) property. The **Execute** method will not return the correct information when used with **adAsyncExecute**, simply because when a command is executed asynchronously, the number of records affected may not yet be known at the time the method returns.

Parameters

Optional. A **Variant** array of parameter values passed with an SQL statement. (Output parameters will not return correct values when passed in this argument.)

Options

Optional. A **Long** value that indicates how the provider should evaluate the [CommandText](#) property of the [Command](#) object. Can be a bitmask value made using [CommandTypeEnum](#) and/or [ExecuteOptionEnum](#) values. For example, you could use both **adCmdText** and **adExecuteNoRecords** together in combination if you want to have ADO evaluate the value of the **CommandText** property as text and indicate that the command should discard and not return any records that might be generated when the command text executes.

Remarks

Using the **Execute** method on a **Command** object executes the query specified in the **CommandText** property of the object. If the **CommandText** property specifies a row-returning query, any results that the execution generates are stored in a new **Recordset** object. If the command is not a row-returning query, the provider returns a closed **Recordset** object. Some application languages allow you to ignore this return value if no **Recordset** is desired.

If the query has parameters, the current values for the **Command** object's parameters are used unless you override these with parameter values passed with the **Execute** call. You can override a subset of the parameters by omitting new values for some of the parameters when calling the **Execute** method. The order in which you specify the parameters is the same order in which the method passes them. For example, if there were four (or more) parameters and you wanted to pass new values for only the first and fourth parameters, you would pass `Array(var1, , , var4)` as the *Parameters* argument.

Note Output parameters will not return correct values when passed in the *Parameters* argument.

An [ExecuteComplete](#) event will be issued when this operation concludes.

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#) | [CommandText Property](#) | [CommandTypeEnum](#) | [Execute Method \(ADO Connection\)](#) | [ExecuteComplete Event](#)

Applies To: [Command Object](#)

© 1998-2002 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Execute Method (ADO Connection)

Executes the specified query, SQL statement, stored procedure, or [provider-specific](#) text.

Syntax

For a non-row-returning command string:

```
connection.Execute CommandText, RecordsAffected, Options
```

For a row-returning command string:

```
Set recordset = connection.Execute (CommandText, RecordsAffected, Op
```

Return Value

Returns a [Recordset](#) object reference.

Parameters

CommandText

A **String** value that contains the SQL statement, stored procedure, a URL, or provider-specific text to execute. Optionally, table names can be used but only if the provider is SQL aware. For example if a table name of "Customers" is used, ADO will automatically prepend the standard SQL Select syntax to form and pass "SELECT * FROM Customers" as a T-SQL statement to the provider.

RecordsAffected

Optional. A **Long** variable to which the provider returns the number of records that the operation affected.

Options

Optional. A **Long** value that indicates how the provider should evaluate the *CommandText* argument. Can be a bitmask of one or more [CommandTypeEnum](#) or [ExecuteOptionEnum](#) values.

Note Use the **ExecuteOptionEnum** value **adExecuteNoRecords** to improve performance by minimizing internal processing.

Do not use the **CommandTypeEnum** values of **adCmdFile** or **adCmdTableDirect** with **Execute**. These values can only be used as options with the [Open](#) and [Requery](#) methods of a **Recordset**.

Remarks

Using the **Execute** method on a [Connection](#) object executes whatever query you pass to the method in the *CommandText* argument on the specified connection. If the *CommandText* argument specifies a row-returning query, any results that the execution generates are stored in a new **Recordset** object. If the command is not intended to return results (for example, an SQL UPDATE query) the provider returns **Nothing** as long as the option **adExecuteNoRecords** is specified; otherwise **Execute** returns a closed **Recordset**.

The returned **Recordset** object is always a read-only, forward-only [cursor](#). If you need a **Recordset** object with more functionality, first create a **Recordset** object with the desired property settings, then use the **Recordset** object's [Open](#) method to execute the query and return the desired cursor type.

The contents of the *CommandText* argument are specific to the provider and can be standard SQL syntax or any special command format that the provider supports.

An [ExecuteComplete](#) event will be issued when this operation concludes.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Execute Method \(ADO Command\)](#) | [ExecuteComplete Event](#)

Applies To: [Connection Object](#)

© 1998-2002 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Find Method

Searches a [Recordset](#) for the row that satisfies the specified criteria. Optionally, the direction of the search, starting row, and offset from the starting row may be specified. If the criteria is met, the current row position is set on the found record; otherwise, the position is set to the end (or start) of the **Recordset**.

Syntax

Find (*Criteria*, *SkipRows*, *SearchDirection*, *Start*)

Parameters

Criteria

A **String** value that contains a statement specifying the column name, comparison operator, and value to use in the search.

SkipRows

Optional. A **Long** value, whose default value is zero, that specifies the row offset from the current row or *Start* bookmark to begin the search. By default, the search will start on the current row.

SearchDirection

Optional. A [SearchDirectionEnum](#) value that specifies whether the search should begin on the current row or the next available row in the direction of the search. An unsuccessful search stops at the end of the **Recordset** if the value is **adSearchForward**. An unsuccessful search stops at the start of the **Recordset** if the value is **adSearchBackward**.

Start

Optional. A **Variant** bookmark that functions as the starting position for the search.

Remarks

Only a single-column name may be specified in *criteria*. This method does not support multi-column searches.

The comparison operator in *Criteria* may be ">" (greater than), "<" (less than),

"=" (equal), ">=" (greater than or equal), "<=" (less than or equal), "<>" (not equal), or "like" (pattern matching).

The value in *Criteria* may be a string, floating-point number, or date. String values are delimited with single quotes or "#" (number sign) marks (for example, "state = 'WA'" or "state = #WA#"). Date values are delimited with "#" (number sign) marks (for example, "start_date > #7/22/97#") and can contain hours, minutes and seconds to indicate time stamps but should not contain milliseconds or errors will occur.

If the comparison operator is "like", the string value may contain an asterisk (*) to find one or more occurrences of any character or substring. For example, "state like 'M*'" matches Maine and Massachusetts. You can also use leading and trailing asterisks to find a substring contained within the values. For example, "state like '*as*'" matches Alaska, Arkansas, and Massachusetts.

Asterisks can be used only at the end of a criteria string, or together at both the beginning and end of a criteria string, as shown above. You cannot use the asterisk as a leading wildcard (*str'), or embedded wildcard (s*r'). This will cause an error.

Note An error will occur if a current row position is not set before calling **Find**. Any method that sets row position, such as [MoveFirst](#), should be called before calling **Find**.

Note If you call the **Find** method on a recordset, and the current position in the recordset is at the last record or end of file (EOF), you will not find anything. You need to call the **MoveFirst** method to set the current position/cursor to the beginning of the recordset.

See Also

[Visual Basic Example](#)

[Index Property](#) | [Optimize Property—Dynamic \(ADO\)](#) | [Seek Method](#)

Applies To: [Recordset Object](#)

ADO 2.5 API Reference

Flush Method

Forces the contents of the [Stream](#) remaining in the ADO buffer to the underlying object with which the **Stream** is associated.

Syntax

Stream.**Flush**

Remarks

This method may be used to send the contents of the stream buffer to the underlying object (for example, the node or file represented by the URL that is the source of the **Stream** object). This method should be called when you want to ensure that all changes made to the contents of a **Stream** have been written. However, with ADO it is not usually necessary to call **Flush**, as ADO continuously flushes its buffer as much as possible in the background. Changes to the content of a **Stream** are made automatically, not cached until **Flush** is called.

Closing a **Stream** with the [Close](#) method flushes the contents of a **Stream** automatically; there is no need to explicitly call **Flush** immediately before **Close**.

See Also

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

GetChildren Method

Returns a [Recordset](#) whose rows represent the children of a collection [Record](#).

Syntax

```
Set recordset = record.GetChildren
```

Return Value

A **Recordset** object for which each row represents a child of the current **Record** object. For example, the children of a **Record** that represents a directory would be the files and subdirectories contained within the parent directory.

Remarks

The provider determines what columns exist in the returned **Recordset**. For example, a document source provider always returns a resource **Recordset**.

See Also

[Example](#) | [Example](#)

Applies To: [Record Object](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

GetChunk Method

Returns all, or a portion, of the contents of a large text or binary data [Field](#) object.

Syntax

```
variable = field.GetChunk( Size )
```

Return Value

Returns a **Variant**.

Parameters

Size

A **Long** expression that is equal to the number of bytes or characters that you want to retrieve.

Remarks

Use the **GetChunk** method on a **Field** object to retrieve part or all of its long binary or character data. In situations where system memory is limited, you can use the **GetChunk** method to manipulate long values in portions, rather than in their entirety.

The data that a **GetChunk** call returns is assigned to *variable*. If *Size* is greater than the remaining data, the **GetChunk** method returns only the remaining data without padding *variable* with empty spaces. If the field is empty, the **GetChunk** method returns a null value.

Each subsequent **GetChunk** call retrieves data starting from where the previous **GetChunk** call left off. However, if you are retrieving data from one field and then you set or read the value of another field in the current record, ADO assumes you have finished retrieving data from the first field. If you call the

GetChunk method on the first field again, ADO interprets the call as a new **GetChunk** operation and starts reading from the beginning of the data. Accessing fields in other [Recordset](#) objects that are not clones of the first **Recordset** object will not disrupt **GetChunk** operations.

If the **adFldLong** bit in the [Attributes](#) property of a **Field** object is set to **True**, you can use the **GetChunk** method for that field.

If there is no current record when you use the **GetChunk** method on a **Field** object, error 3021 (no current record) occurs.

Note The **GetChunk** method does not operate on **Field** objects of a [Record](#) object. It does not perform any operation and will produce a run-time error.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[AppendChunk Method](#) | [Attributes Property](#)

Applies To: [Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

GetRows Method

Retrieves multiple records of a [Recordset](#) object into an array.

Syntax

```
array = recordset.GetRows( Rows, Start, Fields )
```

Return Value

Returns a **VARIANT** whose value is a two-dimensional array.

Parameters

Rows

Optional. A [GetRowsOptionEnum](#) value that indicates the number of records to retrieve. The default is **adGetRowsRest**.

Start

Optional. A **String** value or **VARIANT** that evaluates to the bookmark for the record from which the **GetRows** operation should begin. You can also use a [BookmarkEnum](#) value.

Fields

Optional. A **VARIANT** that represents a single field name or ordinal position, or an array of field names or ordinal position numbers. ADO returns only the data in these fields.

Remarks

Use the **GetRows** method to copy records from a **Recordset** into a two-dimensional array. The first subscript identifies the field and the second identifies the record number. The *array* variable is automatically dimensioned to the correct size when the **GetRows** method returns the data.

If you do not specify a value for the *Rows* argument, the **GetRows** method automatically retrieves all the records in the **Recordset** object. If you request

more records than are available, **GetRows** returns only the number of available records.

If the **Recordset** object supports bookmarks, you can specify at which record the **GetRows** method should begin retrieving data by passing the value of that record's [Bookmark](#) property in the *Start* argument.

If you want to restrict the fields that the **GetRows** call returns, you can pass either a single field name/number or an array of field names/numbers in the *Fields* argument.

After you call **GetRows**, the next unread record becomes the current record, or the [EOF](#) property is set to **True** if there are no more records.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

GetString Method

Returns the [Recordset](#) as a string.

Syntax

```
Variant = recordset.GetString(StringFormat, NumRows, ColumnDelimiter
```

Return Value

Returns the **Recordset** as a string-valued **Variant** (BSTR).

Parameters

StringFormat

A [StringFormatEnum](#) value that specifies how the **Recordset** should be converted to a string. The *RowDelimiter*, *ColumnDelimiter*, and *NullExpr* parameters are used only with a *StringFormat* of **adClipString**.

NumRows

Optional. The number of rows to be converted in the **Recordset**. If *NumRows* is not specified, or if it is greater than the total number of rows in the **Recordset**, then all the rows in the **Recordset** are converted.

ColumnDelimiter

Optional. A delimiter used between columns, if specified, otherwise the TAB character.

RowDelimiter

Optional. A delimiter used between rows, if specified, otherwise the CARRIAGE RETURN character.

NullExpr

Optional. An expression used in place of a null value, if specified, otherwise the empty string.

Remarks

Row data, but no schema data, is saved to the string. Therefore, a **Recordset**

cannot be reopened using this string.

This method is equivalent to the RDO **GetClipString** method.

See Also

[Visual Basic Example](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

LoadFromFile Method

Loads the contents of an existing file into a [Stream](#).

Syntax

```
Stream.LoadFromFile FileName
```

Parameter

FileName

A **String** value that contains the name of a file to be loaded into the **Stream**. *FileName* can contain any valid path and name in UNC format. If the specified file does not exist, a run-time error occurs.

Remarks

This method may be used to load the contents of a local file into a **Stream** object. This may be used to upload the contents of a local file to a server.

The **Stream** object must be already open before calling **LoadFromFile**. This method does not change the binding of the **Stream** object; it will still be bound to the object specified by the URL or **Record** with which the **Stream** was originally opened.

LoadFromFile overwrites the current contents of the **Stream** object with data read from the file. Any existing bytes in the **Stream** are overwritten by the contents of the file. Any previously existing and remaining bytes following the [EOS](#) created by **LoadFromFile**, are truncated.

After a call to **LoadFromFile**, the current position is set to the beginning of the **Stream** ([Position](#) is 0).

Because 2 bytes may be added to the beginning of the stream for encoding, the size of the stream may not exactly match the size of the file from which it was loaded.

See Also

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Move Method

Moves the position of the current record in a [Recordset](#) object.

Syntax

```
recordset.Move NumRecords, Start
```

Parameters

NumRecords

A signed **Long** expression that specifies the number of records that the current record position moves.

Start

Optional. A **String** value or **VARIANT** that evaluates to a bookmark. You can also use a [BookmarkEnum](#) value.

Remarks

The **Move** method is supported on all **Recordset** objects.

If the *NumRecords* argument is greater than zero, the current record position moves forward (toward the end of the **Recordset**). If *NumRecords* is less than zero, the current record position moves backward (toward the beginning of the **Recordset**).

If the **Move** call would move the current record position to a point before the first record, ADO sets the current record to the position before the first record in the recordset (**BOF** is **True**). An attempt to move backward when the **BOF** property is already **True** generates an error.

If the **Move** call would move the current record position to a point after the last record, ADO sets the current record to the position after the last record in the recordset (**EOF** is **True**). An attempt to move forward when the **EOF** property is already **True** generates an error.

Calling the **Move** method from an empty **Recordset** object generates an error.

If you pass the *Start* argument, the move is relative to the record with this bookmark, assuming the **Recordset** object supports bookmarks. If not specified, the move is relative to the current record.

If you are using the [CacheSize](#) property to locally cache records from the [provider](#), passing a *NumRecords* argument that moves the current record position outside the current group of cached records forces ADO to retrieve a new group of records, starting from the destination record. The **CacheSize** property determines the size of the newly retrieved group, and the destination record is the first record retrieved.

If the **Recordset** object is forward only, a user can still pass a *NumRecords* argument less than zero, provided the destination is within the current set of cached records. If the **Move** call would move the current record position to a record before the first cached record, an error will occur. Thus, you can use a record cache that supports full scrolling over a provider that supports only forward scrolling. Because cached records are loaded into memory, you should avoid caching more records than is necessary. Even if a forward-only **Recordset** object supports backward moves in this way, calling the [MovePrevious](#) method on any forward-only **Recordset** object will still generate an error.

Note Support for moving backwards in a forward-only **Recordset** is not predictable, depending upon your provider. If the current record has been positioned after the last record in the **Recordset**, **Move** backwards may not result in the correct current position.

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[MoveFirst, MoveLast, MoveNext, and MovePrevious Methods](#) | [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods \(RDS\)](#) | [MoveRecord Method](#)

Applies To: [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

MoveFirst, MoveLast, MoveNext, and MovePrevious Methods

Moves to the first, last, next, or previous record in a specified [Recordset](#) object and makes that record the current record.

Syntax

```
recordset.{MoveFirst | MoveLast | MoveNext | MovePrevious}
```

Remarks

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**. The **Recordset** object must support bookmarks or backward [cursor](#) movement; otherwise, the method call will generate an error.

A call to either **MoveFirst** or **MoveLast** when the **Recordset** is empty (both **BOF** and **EOF** are **True**) generates an error.

Use the **MoveNext** method to move the current record position one record forward (toward the bottom of the **Recordset**). 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** (**EOF** is **True**). An attempt to move forward when the **EOF** property is already **True** generates an error.

In cases where the **Recordset** has been filtered or sorted and the current record's data is changed, the position may also change. In such cases the **MoveNext** method works normally, but you should be aware that the position is moved one record forward from the new position, not the old position. For example, changing the data in the current record, such that the record is moved to the end of the sorted **Recordset**, would mean that calling **MoveNext** results in ADO setting the current record to the position after the last record in the **Recordset**

(**EOF = True**).

Use the **MovePrevious** method to move the current record position one record backward (toward the top of the **Recordset**). The **Recordset** object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error. 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** (**BOF** is **True**). An attempt to move backward when the **BOF** property is already **True** generates an error. If the **Recordset** object does not support either bookmarks or backward cursor movement, the **MovePrevious** method will generate an error.

If the **Recordset** is forward only and you want to support both forward and backward scrolling, you can use the **CacheSize** property to create a record cache that will support backward cursor movement through the **Move** method. Because cached records are loaded into memory, you should avoid caching more records than is necessary. You can call the **MoveFirst** method in a forward-only **Recordset** object; doing so may cause the provider to re-execute the command that generated the **Recordset** object.

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Move Method](#) | [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods \(RDS\)](#) | [MoveRecord Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

MoveRecord Method

Moves the entity represent by a [Record](#) to another location.

Syntax

```
Record.MoveRecord (Source, Destination, UserName, Password, Options,
```

Parameters

Source

Optional. A **String** value that contains a URL identifying the **Record** to be moved. If *Source* is omitted or specifies an empty string, the object represented by this **Record** is moved. For example, if the **Record** represents a file, the contents of the file are moved to the location specified by *Destination*.

Destination

Optional. A **String** value that contains a URL specifying the location where *Source* will be moved.

UserName

Optional. A **String** value that contains the user ID that, if needed, authorizes access to *Destination*.

Password

Optional. A **String** that contains the password that, if needed, verifies *UserName*.

Options

Optional. A [MoveRecordOptionsEnum](#) value whose default value is **adMoveUnspecified**. Specifies the behavior of this method.

Async

Optional. A **Boolean** value that, when **True**, specifies this operation should be [asynchronous](#).

Return Value

A **String** value. Typically, the value of *Destination* is returned. However, the

exact value returned is provider-dependent.

Remarks

The values of *Source* and *Destination* must not be identical; otherwise, a run-time error occurs. At least the server, path, and resource names must differ.

For files moved using the Internet Publishing Provider, this method updates all hypertext links in files being moved unless otherwise specified by *Options*. This method fails if *Destination* identifies an existing object (for example, a file or directory), unless **adMoveOverWrite** is specified.

Note Use the **adMoveOverWrite** option judiciously. For example, specifying this option when moving a file to a directory will delete the directory and replace it with the file.

Certain attributes of the **Record** object, such as the [ParentURL](#) property, will not be updated after this operation completes. Refresh the **Record** object's properties by closing the **Record**, then re-opening it with the URL of the location where the file or directory was moved.

If this **Record** was obtained from a [Recordset](#), the new location of the moved file or directory will not be reflected immediately in the **Recordset**. Refresh the **Recordset** by closing and re-opening it.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Move Method](#) | [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods](#) | [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods \(RDS\)](#)

Applies To: [Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

NextRecordset Method

Clears the current [Recordset](#) object and returns the next **Recordset** by advancing through a series of commands.

Syntax

```
Set recordset2 = recordset1.NextRecordset( RecordsAffected )
```

Return Value

Returns a **Recordset** object. In the syntax model, *recordset1* and *recordset2* can be the same **Recordset** object, or you can use separate objects. When using separate **Recordset** objects, resetting the **ActiveConnection** property on the original **Recordset** (*recordset1*) after **NextRecordset** has been called will generate an error.

Parameters

RecordsAffected

Optional. A **Long** variable to which the provider returns the number of records that the current operation affected.

Note This parameter only returns the number of records affected by an operation; it does not return a count of records from a select statement used to generate the **Recordset**.

Remarks

Use the **NextRecordset** method to return the results of the next command in a compound command statement or of a stored procedure that returns multiple results. If you open a **Recordset** object based on a compound command statement (for example, "SELECT * FROM table1;SELECT * FROM table2") using the [Execute](#) method on a [Command](#) or the [Open](#) method on a **Recordset**, ADO executes only the first command and returns the results to *recordset*. To

access the results of subsequent commands in the statement, call the **NextRecordset** method.

As long as there are additional results and the **Recordset** containing the compound statements is not [disconnected](#) or [marshaled](#) across process boundaries, the **NextRecordset** method will continue to return **Recordset** objects. If a row-returning command executes successfully but returns no records, the returned **Recordset** object will be open but empty. Test for this case by verifying that the [BOF](#) and [EOF](#) properties are both **True**. If a non-row-returning command executes successfully, the returned **Recordset** object will be closed, which you can verify by testing the [State](#) property on the **Recordset**. When there are no more results, *recordset* will be set to *Nothing*.

The **NextRecordset** method is not available on a disconnected **Recordset** object, where [ActiveConnection](#) has been set to **Nothing** (in Microsoft Visual Basic) or NULL (in other languages).

If an edit is in progress while in immediate update mode, calling the **NextRecordset** method generates an error; call the [Update](#) or [CancelUpdate](#) method first.

To pass parameters for more than one command in the compound statement by filling the [Parameters](#) collection, or by passing an array with the original **Open** or **Execute** call, the parameters must be in the same order in the collection or array as their respective commands in the command series. You must finish reading all the results before reading output parameter values.

Your OLE DB provider determines when each command in a compound statement is executed. The [Microsoft OLE DB Provider for SQL Server](#), for example, executes all commands in a batch upon receiving the compound statement. The resulting **Recordsets** are simply returned when you call **NextRecordset**.

However, other providers may execute the next command in a statement only after **NextRecordset** is called. For these providers, if you explicitly close the **Recordset** object before stepping through the entire command statement, ADO never executes the remaining commands.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Open Method (ADO Connection)

Opens a connection to a data source.

Syntax

```
connection.Open ConnectionString, UserID, Password, Options
```

Parameters

ConnectionString

Optional. A **String** value that contains connection information. See the [ConnectionString](#) property for details on valid settings.

UserID

Optional. A **String** value that contains a user name to use when establishing the connection.

Password

Optional. A **String** value that contains a password to use when establishing the connection.

Options

Optional. A [ConnectOptionEnum](#) value that determines whether this method should return after ([synchronously](#)) or before ([asynchronously](#)) the connection is established.

Remarks

Using the **Open** method on a [Connection](#) object establishes the physical connection to a data source. After this method successfully completes, the connection is live and you can issue commands against it and process the results.

Use the optional *ConnectionString* argument to specify either a connection string containing a series of *argument = value* statements separated by semicolons, or a file or directory resource identified with a URL. The **ConnectionString** property automatically inherits the value used for the *ConnectionString* argument.

Therefore, you can either set the **ConnectionString** property of the **Connection** object before opening it, or use the *ConnectionString* argument to set or override

the current connection parameters during the **Open** method call.

If you pass user and password information both in the *ConnectionString* argument and in the optional *UserID* and *Password* arguments, the *UserID* and *Password* arguments will override the values specified in *ConnectionString*.

When you have concluded your operations over an open **Connection**, use the [Close](#) method to free any associated system resources. Closing an object does not remove it from memory; you can change its property settings and use the **Open** method to open it again later. To completely eliminate an object from memory, set the object variable to *Nothing*.

Remote Data Service Usage When used on a [client-side Connection](#) object, the **Open** method doesn't actually establish a connection to the server until a [Recordset](#) is opened on the **Connection** object.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Open Method \(ADO Record\)](#) | [Open Method \(ADO Recordset\)](#) | [Open Method \(ADO Stream\)](#) | [OpenSchema Method](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Open Method (ADO Record)

Opens an existing [Record](#) object, or creates a new item represented by the **Record** (such as a file or directory).

Syntax

Open *Source*, *ActiveConnection*, *Mode*, *CreateOptions*, *Options*, *UserName*

Parameters

Source

Optional. A **Variant** that may represent the URL of the entity to be represented by this **Record** object, a **Command**, an open [Recordset](#) or another **Record** object, a string containing a SQL SELECT statement or a table name.

ActiveConnection

Optional. A **Variant** that represents the connect string or open [Connection](#) object.

Mode

Optional. A [ConnectModeEnum](#) value, whose default value is **adModeUnknown**, that specifies the access mode for the resultant **Record** object.

CreateOptions

Optional. A [RecordCreateOptionsEnum](#) value, whose default value is **adFailIfNotExists**, that specifies whether an existing file or directory should be opened, or a new file or directory should be created. If set to the default value, the access mode is obtained from the [Mode](#) property. This parameter is ignored when the *Source* parameter doesn't contain a URL.

Options

Optional. A [RecordOpenOptionsEnum](#) value, whose default value is **adOpenRecordUnspecified**, that specifies options for opening the **Record**. These values may be combined.

UserName

Optional. A **String** value that contains the user ID that, if needed, authorizes access to *Source*.

Password

Optional. A **String** value that contains the password that, if needed, verifies *UserName*.

Remarks

Source may be:

- A URL. If the protocol for the URL is http, then the Internet Provider will be invoked by default. If the URL points to a node that contains an executable script (such as an .ASP page), then a **Record** containing the source rather than the executed contents is opened by default. Use the *Options* argument to modify this behavior.
- A **Record** object. A **Record** object opened from another **Record** will clone the original **Record** object.
- A **Command** object. The opened **Record** object represents the single row returned by executing the **Command**. If the results contain more than a single row, the contents of the first row are placed in the record and an error may be added to the **Errors** collection.
- A SQL SELECT statement. The opened **Record** object represents the single row returned by executing the contents of the string. If the results contain more than a single row, the contents of the first row are placed in the record and an error may be added to the **Errors** collection.
- A table name.

If the **Record** object represents an entity that cannot be accessed with a URL (for example, a row of a **Recordset** derived from a database), then the values of both the [ParentURL](#) property and the field accessed with the **adRecordURL** constant are null.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Example](#) | [Example](#)

[Open Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#) | [Open](#)

[Method \(ADO Stream\)](#) | [OpenSchema Method](#)

Applies To: [Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Open Method (ADO Recordset)

Opens a [cursor](#).

Syntax

```
recordset.Open Source, ActiveConnection, CursorType, LockType, Options
```

Parameters

Source

Optional. A **Variant** that evaluates to a valid [Command](#) object, an SQL statement, a table name, a stored procedure call, a [URL](#), or the name of a file or [Stream](#) object containing a persistently stored [Recordset](#).

ActiveConnection

Optional. Either a **Variant** that evaluates to a valid [Connection object variable](#) name, or a **String** that contains [ConnectionString](#) parameters.

CursorType

Optional. A [CursorTypeEnum](#) value that determines the type of cursor that the [provider](#) should use when opening the **Recordset**. The default value is **adOpenForwardOnly**.

LockType

Optional. A [LockTypeEnum](#) value that determines what type of locking (concurrency) the provider should use when opening the **Recordset**. The default value is **adLockReadOnly**.

Options

Optional. A **Long** value that indicates how the provider should evaluate the *Source* argument if it represents something other than a **Command** object, or that the **Recordset** should be restored from a file where it was previously saved. Can be one or more [CommandTypeEnum](#) or [ExecuteOptionEnum](#) values, which can be combined with a bitwise AND operator.

Note If you open a **Recordset** from a **Stream** containing a persisted **Recordset**, using an **ExecuteOptionEnum** value of **adAsyncFetchNonBlocking** will not have an effect; the fetch will be synchronous and blocking.

The **ExecuteOpenEnum** values of **adExecuteNoRecords** or **adExecuteStream** should not be used with **Open**.

Remarks

The default cursor for an ADO **Recordset** is a forward-only, read-only cursor located on the server.

Using the **Open** method on a **Recordset** object opens a cursor that represents records from a base table, the results of a query, or a previously saved **Recordset**.

Use the optional *Source* argument to specify a data source using one of the following: a **Command** object variable, an SQL statement, a stored procedure, a table name, a URL, or a complete file path name. If *Source* is a file path name, it can be a full path ("c:\dir\file.rst"), a relative path ("..\file.rst"), or a URL ("http://files/file.rst").

It is not a good idea to use the *Source* argument of the **Open** method to perform an action query that doesn't return records because there is no easy way to determine whether the call succeeded. The **Recordset** returned by such a query will be closed. Call the [Execute](#) method of a **Command** object or the [Execute](#) method of a **Connection** object instead to perform a query that, such as a SQL INSERT statement, that doesn't return records.

The *ActiveConnection* argument corresponds to the [ActiveConnection](#) property and specifies in which connection to open the **Recordset** object. If you pass a connection definition for this argument, ADO opens a new connection using the specified parameters. After opening the **Recordset** with a client-side cursor (**CursorLocation** = **adUseClient**), you can change the value of this property to send updates to another provider. Or you can set this property to **Nothing** (in Microsoft Visual Basic) or NULL to disconnect the **Recordset** from any provider. Changing **ActiveConnection** for a server-side cursor generates an error, however.

For the other arguments that correspond directly to properties of a **Recordset** object (*Source*, *CursorType*, and *LockType*), the relationship of the arguments to the properties is as follows:

- The property is read/write before the **Recordset** object is opened.
- The property settings are used unless you pass the corresponding arguments when executing the **Open** method. If you pass an argument, it overrides the corresponding property setting, and the property setting is updated with the argument value.
- After you open the **Recordset** object, these properties become read-only.

Note The **ActiveConnection** property is read only for **Recordset** objects whose [Source](#) property is set to a valid **Command** object, even if the **Recordset** object isn't open.

If you pass a **Command** object in the *Source* argument and also pass an *ActiveConnection* argument, an error occurs. The **ActiveConnection** property of the **Command** object must already be set to a valid **Connection** object or connection string.

If you pass something other than a **Command** object in the *Source* argument, you can use the *Options* argument to optimize evaluation of the *Source* argument. If the *Options* argument is not defined, you may experience diminished performance because ADO must make calls to the provider to determine if the argument is an SQL statement, a stored procedure, a URL, or a table name. If you know what *Source* type you're using, setting the *Options* argument instructs ADO to jump directly to the relevant code. If the *Options* argument does not match the *Source* type, an error occurs.

If you pass a **Stream** object in the *Source* argument, you should not pass information into the other arguments. Doing so will generate an error. The **ActiveConnection** information is not retained when a **Recordset** is opened from a **Stream**.

The default for the *Options* argument is **adCmdFile** if no connection is associated with the **Recordset**. This will typically be the case for persistently stored **Recordset** objects.

If the data source returns no records, the provider sets both the [BOF](#) and [EOF](#) properties to **True**, and the current record position is undefined. You can still add new data to this empty **Recordset** object if the cursor type allows it.

When you have concluded your operations over an open **Recordset** object, use the [Close](#) method to free any associated system resources. Closing an object

does not remove it from memory; you can change its property settings and use the **Open** method to open it again later. To completely eliminate an object from memory, set the object variable to *Nothing*.

Before the **ActiveConnection** property is set, call **Open** with no operands to create an instance of a **Recordset** created by appending fields to the **Recordset Fields** collection.

If you have set the [CursorLocation](#) property to **adUseClient**, you can retrieve rows asynchronously in one of two ways. The recommended method is to set *Options* to **adAsyncFetch**. Alternatively, you can use the "Asynchronous Rowset Processing" [dynamic property](#) in the [Properties](#) collection, but related retrieved events can be lost if you do not set the **Options** parameter to **adAsyncFetch**.

Note Background fetching in the MS Remote provider is supported only through the **Open** method's *Options* parameter.

Note URLs using the http scheme will automatically invoke the [Microsoft OLE DB Provider for Internet Publishing](#). For more information, see [Absolute and Relative URLs](#).

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#) | [Visual Basic Example](#)

[Open Method \(ADO Connection\)](#) | [Open Method \(ADO Record\)](#) | [Open Method \(ADO Stream\)](#) | [OpenSchema Method](#) | [Save Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Open Method (ADO Stream)

Opens a [Stream](#) object to manipulate streams of binary or text data.

Syntax

```
Stream.Open Source, Mode, OpenOptions, UserName, Password
```

Parameters

Source

Optional. A **VARIANT** value that specifies the source of data for the **Stream**. *Source* may contain an [absolute URL](#) string that points to an existing node in a well-known tree structure, like an e-mail or file system. A URL should be specified using the URL keyword ("URL=scheme://server/folder"). Alternately, *Source* may contain a reference to an already open [Record](#) object, which opens the default stream associated with the **Record**. If *Source* is not specified, a **Stream** is instantiated and opened, associated with no underlying source by default. For more information about URL schemes and their associated providers, see [Absolute and Relative URLs](#).

Mode

Optional. A [ConnectModeEnum](#) value that specifies the access mode for the resultant **Stream** (for example, read/write or read-only). Default value is **adModeUnknown**. See the [Mode](#) property for more information about access modes. If *Mode* is not specified, it is inherited by the source object. For example, if the source **Record** is opened in read-only mode, the **Stream** will also be opened in read-only mode by default.

OpenOptions

Optional. A [StreamOpenOptionsEnum](#) value. Default value is **adOpenStreamUnspecified**.

UserName

Optional. A **String** value that contains the user identification that, if needed, accesses the **Stream** object.

Password

Optional. A **String** value that contains the password that, if needed, accesses the **Stream** object.

Remarks

When a **Record** object is passed in as the source parameter, the *UserID* and *Password* parameters are not used because access to the **Record** object is already available. Similarly, the [Mode](#) of the **Record** object is transferred to the **Stream** object. When *Source* is not specified, the **Stream** opened contains no data and has a [Size](#) of zero (0). To avoid losing any data that is written to this **Stream** when the **Stream** is closed, save the **Stream** with the [CopyTo](#) or [SaveToFile](#) methods, or save it to another memory location.

An *OpenOptions* value of **adOpenStreamFromRecord** identifies the contents of the *Source* parameter to be an already open **Record** object. The default behavior is to treat *Source* as a URL that points directly to a node in a tree structure, such as a file. The [default stream](#) associated with that node is opened.

While the **Stream** is not open, it is possible to read all the read-only properties of the **Stream**. If a **Stream** is opened asynchronously, all subsequent operations (other than checking the [State](#) and other read-only properties) are blocked until the **Open** operation is completed.

In addition to the options discussed above, by not specifying *Source*, you can simply instantiate a **Stream** object in memory without associating it with an underlying source. You can dynamically add data to the stream simply by writing binary or text data to the **Stream** with [Write](#) or [WriteText](#), or by loading data from a file with [LoadFromFile](#).

See Also

[Example](#)

[Open Method \(ADO Connection\)](#) | [Open Method \(ADO Record\)](#) | [Open Method \(ADO Recordset\)](#) | [OpenSchema Method](#) | [SaveToFile Method](#)

Applies To: [Stream Object](#)

ADO 2.5 API Reference

OpenSchema Method

Obtains database schema information from the [provider](#).

Syntax

```
Set recordset = connection.OpenSchema (QueryType, Criteria, SchemaID)
```

Return Values

Returns a [Recordset](#) object that contains schema information. The **Recordset** will be opened as a read-only, static [cursor](#). The *QueryType* determines what columns appear in the **Recordset**.

Parameters

QueryType

Any [SchemaEnum](#) value that represents the type of schema query to run.

Criteria

Optional. An array of query constraints for each *QueryType* option, as listed in **SchemaEnum**.

SchemaID

The GUID for a provider-schema query not defined by the OLE DB specification. This parameter is required if *QueryType* is set to **adSchemaProviderSpecific**; otherwise, it is not used.

Remarks

The **OpenSchema** method returns self-descriptive information about the data source, such as what tables are in the data source, the columns in the tables, and the data types supported.

The *QueryType* argument is a GUID that indicates the columns (schemas) returned. The OLE DB specification has a full list of schemas.

The *Criteria* argument limits the results of a schema query. *Criteria* specifies an array of values that must occur in a corresponding subset of columns, called *constraint columns*, in the resulting **Recordset**.

The constant **adSchemaProviderSpecific** is used for the *QueryType* argument if the provider defines its own nonstandard schema queries outside those listed above. When this constant is used, the *SchemaID* argument is required to pass the GUID of the schema query to execute. If *QueryType* is set to **adSchemaProviderSpecific** but *SchemaID* is not provided, an error will result.

Providers are not required to support all of the OLE DB standard schema queries. Specifically, only **adSchemaTables**, **adSchemaColumns**, and **adSchemaProviderTypes** are required by the OLE DB specification. However, the provider is not required to support the *Criteria* constraints listed above for those schema queries.

Remote Data Service Usage The **OpenSchema** method is not available on a client-side [Connection](#) object.

Note In Visual Basic, columns that have a four-byte unsigned integer (DBTYPE UI4) in the **Recordset** returned from the **OpenSchema** method on the **Connection** object cannot be compared to other variables. For more information about OLE DB data types, see Chapter 13 and Appendix A of the *Microsoft OLE DB Programmer's Reference*.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Open Method \(ADO Connection\)](#) | [Open Method \(ADO Record\)](#) | [Open Method \(ADO Recordset\)](#) | [Open Method \(ADO Stream\)](#) | [Appendix A: Providers](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Read Method

Reads a specified number of bytes from a binary [Stream](#) object.

Syntax

```
Variant = Stream.Read ( NumBytes )
```

Parameters

NumBytes

Optional. A **Long** value that specifies the number of bytes to read from the file or the [StreamReadEnum](#) value **adReadAll**, which is the default.

Return Value

The **Read** method reads a specified number of bytes or the entire stream from a **Stream** object and returns the resulting data as a **Variant**.

Remarks

If *NumBytes* is more than the number of bytes left in the **Stream**, only the bytes remaining are returned. The data read is not padded to match the length specified by *NumBytes*. If there are no bytes left to read, a variant with a null value is returned. **Read** cannot be used to read backwards.

Note *NumBytes* always measures bytes. For text **Stream** objects ([Type](#) is **adTypeText**), use [ReadText](#).

See Also

[Example](#)

[ReadText Method](#)

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ReadText Method

Reads specified number of characters from a text [Stream](#) object.

Syntax

```
String = Stream.ReadText ( NumChars )
```

Parameters

NumChars

Optional. A **Long** value that specifies the number of characters to read from the file, or a [StreamReadEnum](#) value. The default value is **adReadAll**.

Return Value

The **ReadText** method reads a specified number of characters, an entire line, or the entire stream from a **Stream** object and returns the resulting string.

Remarks

If *NumChar* is more than the number of characters left in the stream, only the characters remaining are returned. The string read is not padded to match the length specified by *NumChar*. If there are no characters left to read, a variant whose value is null is returned. **ReadText** cannot be used to read backwards.

Note The **ReadText** method is used with text streams ([Type](#) is **adTypeText**). For binary streams (**Type** is **adTypeBinary**), use [Read](#).

See Also

[Example](#)

[Read Method](#)

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Refresh Method

Updates the objects in a collection to reflect objects available from, and specific to, the [provider](#).

Syntax

```
collection.Refresh
```

Remarks

The **Refresh** method accomplishes different tasks depending on the collection from which you call it.

Parameters

Using the **Refresh** method on a [Command](#) object's [Parameters](#) collection retrieves provider-side parameter information for the stored procedure or [parameterized](#) query specified in the **Command** object. The collection will be empty for providers that do not support stored procedure calls or parameterized queries.

You should set the [ActiveConnection](#) property of the **Command** object to a valid [Connection](#) object, the [CommandText](#) property to a valid command, and the [CommandType](#) property to **adCmdStoredProc** before calling the **Refresh** method.

If you access the **Parameters** collection before calling the **Refresh** method, ADO will automatically call the method and populate the collection for you.

Note If you use the **Refresh** method to obtain parameter information from the provider and it returns one or more variable-length data type [Parameter](#) objects, ADO may allocate memory for the parameters based on their maximum potential size, which will cause an error during execution. You should explicitly set the [Size](#) property for these parameters before calling the [Execute](#) method to prevent errors.

Fields

Using the **Refresh** method on the **Fields** collection has no visible effect. To retrieve changes from the underlying database structure, you must use either the [Requery](#) method or, if the [Recordset](#) object does not support bookmarks, the [MoveFirst](#) method.

Properties

Using the **Refresh** method on a **Properties** collection of some objects populates the collection with the [dynamic properties](#) that the [provider](#) exposes. These properties provide information about functionality specific to the provider, beyond the built-in properties ADO supports.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual VJ++ Example](#)

[Count Property](#) | [Refresh Method \(RDS\)](#)

Applies To: [Axes Collection](#) | [Columns Collection](#) | [CubeDefs Collection](#) | [Dimensions Collection](#) | [Errors Collection](#) | [Fields Collection](#) | [Groups Collection](#) | [Hierarchies Collection](#) | [Indexes Collection](#) | [Keys Collection](#) | [Levels Collection](#) | [Members Collection](#) | [Parameters Collection](#) | [Positions Collection](#) | [Procedures Collection](#) | [Properties Collection](#) | [Tables Collection](#) | [Users Collection](#) | [Views Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Requery Method

Updates the data in a [Recordset](#) object by re-executing the query on which the object is based.

Syntax

```
recordset.Requery Options
```

Parameter

Options

Optional. A bitmask that contains [ExecuteOptionEnum](#) and [CommandTypeEnum](#) values affecting this operation.

Note If *Options* is set to **adAsyncExecute**, this operation will execute [asynchronously](#) and a [RecordsetChangeComplete](#) event will be issued when it concludes.

The **ExecuteOpenEnum** values of **adExecuteNoRecords** or **adExecuteStream** should not be used with **Requery**.

Remarks

Use the **Requery** method to refresh the entire contents of a **Recordset** object from the data source by reissuing the original command and retrieving the data a second time. Calling this method is equivalent to calling the [Close](#) and [Open](#) methods in succession. If you are editing the current record or adding a new record, an error occurs.

While the **Recordset** object is open, the properties that define the nature of the [cursor](#) ([CursorType](#), [LockType](#), [MaxRecords](#), and so forth) are read-only. Thus, the **Requery** method can only refresh the current cursor. To change any of the cursor properties and view the results, you must use the [Close](#) method so that the properties become read/write again. You can then change the property settings and call the [Open](#) method to reopen the cursor.

See Also

[Visual Basic Example](#) | [VBScript Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[CommandText Property](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Resync Method

Refreshes the data in the current [Recordset](#) object, or [Fields](#) collection of a [Record](#) object, from the underlying database.

Syntax

```
Recordset.Resync AffectRecords, ResyncValues  
Record.Fields.Resync ResyncValues
```

Parameters

AffectRecords

Optional. An [AffectEnum](#) value that determines how many records the **Resync** method will affect. The default value is **adAffectAll**. This value is not available with the **Resync** method of the **Fields** collection of a **Record** object.

ResyncValues

Optional. A [ResyncEnum](#) value that specifies whether underlying values are overwritten. The default value is **adResyncAllValues**.

Remarks

Recordset

Use the **Resync** method to resynchronize records in the current **Recordset** with the underlying database. This is useful if you are using either a static or forward-only [cursor](#), but you want to see any changes in the underlying database.

If you set the [CursorLocation](#) property to **adUseClient**, **Resync** is only available for non-read-only **Recordset** objects.

Unlike the [Requery](#) method, the **Resync** method does not re-execute the **Recordset** object's underlying command. New records in the underlying database will not be visible.

If the attempt to resynchronize fails because of a conflict with the underlying data (for example, a record has been deleted by another user), the provider returns warnings to the [Errors](#) collection and a run-time error occurs. Use the [Filter](#) property (**adFilterConflictingRecords**) and the [Status](#) property to locate records with conflicts.

If the [Unique Table](#) and [Resync Command](#) dynamic properties are set, and the **Recordset** is the result of executing a JOIN operation on multiple tables, then the **Resync** method will execute the command given in the **Resync Command** property only on the table named in the **Unique Table** property.

Fields

Use the **Resync** method to resynchronize the values of the **Fields** collection of a **Record** object with the underlying data source. The [Count](#) property is not affected by this method.

If *ResyncValues* is set to **adResyncAllValues** (the default value), then the [UnderlyingValue](#), [Value](#), and [OriginalValue](#) properties of [Field](#) objects in the collection are synchronized. If *ResyncValues* is set to **adResyncUnderlyingValues**, only the **UnderlyingValue** property is synchronized.

The value of the **Status** property for each **Field** object at the time of the call also affects the behavior of **Resync**. For **Field** objects with **Status** values of **adFieldPendingUnknown** or **adFieldPendingInsert**, **Resync** has no effect. For **Status** values of **adFieldPendingChange** or **adFieldPendingDelete**, **Resync** synchronizes data values for fields that still exist at the data source.

Resync will not modify **Status** values of **Field** objects unless an error occurs when **Resync** is called. For example, if the field no longer exists, the provider will return an appropriate **Status** value for the **Field** object, such as **adFieldDoesNotExist**. Returned **Status** values may be logically combined within the value of the **Status** property.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Clear Method](#) | [UnderlyingValue Property](#)

Applies To: [Fields Collection](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Save Method

Saves the [Recordset](#) in a file or [Stream](#) object.

Syntax

```
recordset.Save Destination, PersistFormat
```

Parameters

Destination

Optional. A **Variant** that represents the complete path name of the file where the **Recordset** is to be saved, or a reference to a **Stream** object.

PersistFormat

Optional. A [PersistFormatEnum](#) value that specifies the format in which the **Recordset** is to be saved (XML or ADTG). The default value is **adPersistADTG**.

Remarks

The **Save** method can only be invoked on an open **Recordset**. Use the [Open](#) method to later restore the **Recordset** from *Destination*.

If the [Filter](#) property is in effect for the **Recordset**, then only the rows accessible under the filter are saved. If the **Recordset** is hierarchical, then the current [child Recordset](#) and its children are saved, including the [parent Recordset](#). If the **Save** method of a child **Recordset** is called, the child and all its children are saved, but the parent is not.

The first time you save the **Recordset**, it is optional to specify *Destination*. If you omit *Destination*, a new file will be created with a name set to the value of the [Source](#) property of the **Recordset**.

Omit *Destination* when you subsequently call **Save** after the first save, or a run-time error will occur. If you subsequently call **Save** with a new *Destination*, the **Recordset** is saved to the new destination. However, the new destination and the

original destination will both be open.

Save does not close the **Recordset** or *Destination*, so you can continue to work with the **Recordset** and save your most recent changes. *Destination* remains open until the **Recordset** is closed.

For reasons of security, the **Save** method permits only the use of low and custom security settings from a script executed by Microsoft Internet Explorer. For a more detailed explanation of security issues, see "ADO and RDS Security Issues in Microsoft Internet Explorer" under ActiveX Data Objects (ADO) Technical Articles in Microsoft Data Access Technical Articles.

If the **Save** method is called while an asynchronous **Recordset** fetch, execute, or update operation is in progress, then **Save** waits until the [asynchronous](#) operation is complete.

Records are saved beginning with the first row of the **Recordset**. When the **Save** method is finished, the current row position is moved to the first row of the **Recordset**.

For best results, set the [CursorLocation](#) property to **adUseClient** with **Save**. If your [provider](#) does not support all of the functionality necessary to save **Recordset** objects, the Cursor Service will provide that functionality.

When a **Recordset** is persisted with the **CursorLocation** property set to **adUseServer**, the update capability for the **Recordset** is limited. Typically, only single-table updates, insertions, and deletions are allowed (dependant upon provider functionality). The [Resync](#) method is also unavailable in this configuration.

Note Saving a **Recordset** with **Fields** of type **adVariant**, **adIDispatch**, or **adIUnknown** is not supported by ADO and can cause unpredictable results.

Only **Filters** in the form of Criteria Strings (e.g. OrderDate > '12/31/1999') affect the contents of a persisted **Recordset**. Filters created with an Array of **Bookmarks** or using a value from the **FilterGroupEnum** will not affect the contents of the persisted **Recordset**. These rules apply to **Recordsets** created with either client-side or server-side cursors.

Because the *Destination* parameter can accept any object that supports the OLE DB IStream interface, you can save a **Recordset** directly to the ASP Response object. For more details, please see the [XML Recordset Persistence Scenario](#).

You can also save a **Recordset** in XML format to an instance of an MSXML DOM object, as is shown in the following Visual Basic code:

```
Dim xDOM As New MSXML.DOMDocument
Dim rsXML As New ADODB.Recordset
Dim sSQL As String, sConn As String

sSQL = "SELECT customerid, companyname, contactname FROM customers"
sConn="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\Program Files
      "\Common Files\System\msadc\samples\NWind.mdb"
rsXML.Open sSQL, sConn
rsXML.Save xDOM, adPersistXML 'Save Recordset directly into a DOM
...
```

Note Two limitations apply when saving hierarchical **Recordsets** (data shapes) in XML format. You cannot save into XML if the hierarchical **Recordset** contains pending updates, and you cannot save a parameterized hierarchical **Recordset**.

A Recordset saved in XML format is saved using UTF-8 format. When such a file is loaded into an ADO Stream, the Stream object will not attempt to open a Recordset from the stream unless the Charset property of the stream is set to the appropriate value for UTF-8 format.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[Open Method \(ADO Recordset\)](#) | [Open Method \(ADO Stream\)](#) | [SaveToFile Method](#)

Applies To: [Recordset Object](#) | [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

SaveToFile Method

Saves the binary contents of a [Stream](#) to a file.

Syntax

```
Stream.SaveToFile FileName, SaveOptions
```

Parameters

FileName

A **String** value that contains the fully-qualified name of the file to which the contents of the **Stream** will be saved. You can save to any valid local location, or any location you have access to via a UNC value.

SaveOptions

A [SaveOptionsEnum](#) value that specifies whether a new file should be created by **SaveToFile**, if it does not already exist. Default value is **adSaveCreateNotExists**. With these options you can specify that an error occurs if the specified file does not exist. You can also specify that **SaveToFile** overwrites the current contents of an existing file.

Note If you overwrite an existing file (when **adSaveCreateOverwrite** is set), **SaveToFile** truncates any bytes from the original existing file that follow the new [EOS](#).

Remarks

SaveToFile may be used to copy the contents of a **Stream** object to a local file. There is no change in the contents or properties of the **Stream** object. The **Stream** object must be open before calling **SaveToFile**.

This method does not change the association of the **Stream** object to its underlying source. The **Stream** object will still be associated with the original [URL](#) or **Record** that was its source when opened.

After a **SaveToFile** operation, the current position ([Position](#)) in the stream is set

to the beginning of the stream (0).

See Also

[Open Method \(ADO Stream\)](#) | [Save Method](#)

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Seek Method

Searches the index of a [Recordset](#) to quickly locate the row that matches the specified values, and changes the current row position to that row.

Syntax

```
recordset.Seek KeyValues, SeekOption
```

Parameters

KeyValues

An array of **Variant** values. An index consists of one or more columns and the array contains a value to compare against each corresponding column.

SeekOption

A [SeekEnum](#) value that specifies the type of comparison to be made between the columns of the index and the corresponding *KeyValues*.

Remarks

Use the **Seek** method in conjunction with the [Index](#) property if the underlying [provider](#) supports indexes on the **Recordset** object. Use the [Supports\(adSeek\)](#) method to determine whether the underlying provider supports **Seek**, and the [Supports\(adIndex\)](#) method to determine whether the provider supports indexes. (For example, the [OLE DB Provider for Microsoft Jet](#) supports **Seek** and **Index**.)

If **Seek** does not find the desired row, no error occurs, and the row is positioned at the end of the **Recordset**. Set the **Index** property to the desired index before executing this method.

This method is supported only with server-side cursors. **Seek** is not supported when the **Recordset** object's [CursorLocation](#) property value is **adUseClient**.

This method can only be used when the **Recordset** object has been opened with a [CommandTypeEnum](#) value of **adCmdTableDirect**.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#)

[Find Method](#) | [Index Property](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

SetEOS Method

Sets the position that is the end of the stream.

Syntax

Stream.**SetEOS**

Remarks

SetEOS updates the value of the [EOS](#) property, by making the current [Position](#) the end of the stream. Any bytes or characters following the current position are truncated.

Since [Write](#), [WriteText](#), and [CopyTo](#) do not truncate any extra values in existing **Stream** objects, you can truncate these bytes or characters by setting the new end-of-stream position with **SetEOS**.

Caution If you set **EOS** to a position before the actual end of the stream, you will lose all data after the new **EOS** position.

See Also

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

SkipLine Method

Skips one entire line when reading a text stream.

Syntax

Stream.SkipLine

Remarks

All characters up to, and including the next line separator, are skipped. By default, the [LineSeparator](#) is **adCRLF**. If you attempt to skip past [EOS](#), the current position will simply remain at **EOS**.

The **SkipLine** method is used with text streams ([Type](#) is **adTypeText**).

See Also

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Stat Method

Retrieves information about a **Stream** object.

Syntax

Long *stream*.Stat(StatStg, StatFlag)

Return Value

A long value indicating the status of the operation.

Parameters

StatStg

A STATSTG structure that will be filled in with information about the stream. The implementation of the Stat method used by the ADO Stream object does not fill in all of the fields of the structure.

StatFlag

Specifies that this method does not return some of the members in the STATSTG structure, thus saving a memory allocation operation. Values are taken from the STATFLAG enumeration.

The STATFLAG enumeration has two values

Constant	Value
STATFLAG_DEFAULT	0
STATFLAG_NONAME	1

Remarks

The version of the Stat method implemented for the ADO Stream object fills in the following fields of the STATSTG structure:

pwcsName

A string containing the name of the stream, if one is available and the

StatFlag value STATFLAG_NONAME was not specified.

cbSize

Specifies the size in bytes of the stream or byte array.

mtime

Indicates the last modification time for this storage, stream, or byte array.

ctime

Indicates the creation time for this storage, stream, or byte array.

atime

Indicates the last access time for this storage, stream or byte array.

If STATFLAG_NONAME is specified in the StatFlag parameter, the name of the stream is not returned.

If STATFLAG_NONAME was not specified in the StatFlag parameter, and there is no name available for the current stream, this value will be E_NOTIMPL.

See Also

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Supports Method

Determines whether a specified [Recordset](#) object supports a particular type of functionality.

Syntax

```
boolean = recordset.Supports( CursorOptions )
```

Return Value

Returns a **Boolean** value that indicates whether all of the features identified by the *CursorOptions* argument are supported by the [provider](#).

Parameters

CursorOptions

A **Long** expression that consists of one or more [CursorOptionEnum](#) values.

Remarks

Use the **Supports** method to determine what types of functionality a **Recordset** object supports. If the **Recordset** object supports the features whose corresponding constants are in *CursorOptions*, the **Supports** method returns **True**. Otherwise, it returns **False**.

Note Although the **Supports** method may return **True** for a given functionality, it does not guarantee that the provider can make the feature available under all circumstances. The **Supports** method simply returns whether the provider can support the specified functionality, assuming certain conditions are met. For example, the **Supports** method may indicate that a **Recordset** object supports updates even though the cursor is based on a multiple table join, some columns of which are not updatable.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[CursorType Property](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Update Method

Saves any changes you make to the current row of a [Recordset](#) object, or the [Fields](#) collection of a [Record](#) object.

Syntax

```
recordset.Update Fields, Values  
record.Fields.Update
```

Parameters

Fields

Optional. A **Variant** that represents a single name, or a **Variant** array that represents names or ordinal positions of the field or fields you wish to modify.

Values

Optional. A **Variant** that represents a single value, or a **Variant** array that represents values for the field or fields in the new record.

Remarks

Recordset

Use the **Update** method to save any changes you make to the current record of a **Recordset** object since calling the [AddNew](#) method or since changing any field values in an existing record. The **Recordset** object must support updates.

To set field values, do one of the following:

- Assign values to a [Field](#) object's [Value](#) property and call the **Update** method.
- Pass a field name and a value as arguments with the **Update** call.
- Pass an array of field names and an array of values with the **Update** call.

When you use arrays of fields and values, there must be an equal number of

elements in both arrays. Also, the order of field names must match the order of field values. If the number and order of fields and values do not match, an error occurs.

If the **Recordset** object supports batch updating, you can cache multiple changes to one or more records locally until you call the [UpdateBatch](#) method. If you are editing the current record or adding a new record when you call the **UpdateBatch** method, ADO will automatically call the **Update** method to save any pending changes to the current record before transmitting the batched changes to the provider.

If you move from the record you are adding or editing before calling the **Update** method, ADO will automatically call **Update** to save the changes. You must call the [CancelUpdate](#) method if you want to cancel any changes made to the current record or discard a newly added record.

The current record remains current after you call the **Update** method.

Record

The **Update** method finalizes additions, deletions, and updates to fields in the [Fields](#) collection of a **Record** object.

For example, fields deleted with the **Delete** method are marked for deletion immediately but remain in the collection. The **Update** method must be called to actually delete these fields from the provider's collection.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#)

[AddNew Method](#) | [CancelUpdate Method](#) | [EditMode Property](#) | [UpdateBatch Method](#)

Applies To: [Fields Collection](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

UpdateBatch Method

Writes all pending batch updates to disk.

Syntax

```
recordset.UpdateBatch AffectRecords
```

Parameters

AffectRecords

Optional. An [AffectEnum](#) value that indicates how many records the **UpdateBatch** method will affect.

Remarks

Use the **UpdateBatch** method when modifying a **Recordset** object in batch update mode to transmit all changes made in a **Recordset** object to the underlying database.

If the **Recordset** object supports batch updating, you can cache multiple changes to one or more records locally until you call the **UpdateBatch** method. If you are editing the current record or adding a new record when you call the **UpdateBatch** method, ADO will automatically call the [Update](#) method to save any pending changes to the current record before transmitting the batched changes to the provider. You should use batch updating with either a keyset or static cursor only.

Note Specifying **adAffectGroup** as the value for this parameter will result in an error when there are no visible records in the current **Recordset** (such as a filter for which no records match).

If the attempt to transmit changes fails for any or all records because of a conflict with the underlying data (for example, a record has already been deleted by another user), the provider returns warnings to the [Errors](#) collection and a run-time error occurs. Use the [Filter](#) property (**adFilterAffectedRecords**) and

the [Status](#) property to locate records with conflicts.

To cancel all pending batch updates, use the [CancelBatch](#) method.

If the [Unique Table](#) and [Update Resync dynamic properties](#) are set, and the **Recordset** is the result of executing a JOIN operation on multiple tables, then the execution of the **UpdateBatch** method is implicitly followed by the [Resync](#) method depending on the settings of the **Update Resync** property.

The order in which the individual updates of a batch are performed on the data source is not necessarily the same as the order in which they were performed on the local **Recordset**. Update order is dependent upon the provider. Take this into account when coding updates that are related to one another, such as foreign key constraints on an insert or update.

See Also

[Visual Basic Example](#) | [Visual C++ Example](#) | [Visual J++ Example](#)

[CancelBatch Method](#) | [Clear Method](#) | [LockType Property](#) | [Update Method](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Write Method

Writes binary data to a [Stream](#) object.

Syntax

```
Stream.Write Buffer
```

Parameters

Buffer

A **Variant** that contains an array of bytes to be written.

Remarks

Specified bytes are written to the **Stream** object without any intervening spaces between each byte.

The current [Position](#) is set to the byte following the written data. The **Write** method does not truncate the rest of the data in a stream. If you want to truncate these bytes, call [SetEOS](#).

If you write past the current [EOS](#) position, the [Size](#) of the **Stream** will be increased to contain any new bytes, and **EOS** will move to the new last byte in the **Stream**.

Note The **Write** method is used with binary streams ([Type](#) is **adTypeBinary**). For text streams (**Type** is **adTypeText**), use [WriteText](#).

See Also

[WriteText Method](#)

Applies To: [Stream Object](#)

ADO 2.5 API Reference

WriteText Method

Writes a specified text string to a [Stream](#) object.

Syntax

```
Stream.WriteText Data, Options
```

Parameters

Data

A **String** value that contains the text in characters to be written.

Options

Optional. A [StreamWriteEnum](#) value that specifies whether a line separator character must be written at the end of the specified string.

Remarks

Specified strings are written to the **Stream** object without any intervening spaces or characters between each string.

The current [Position](#) is set to the character following the written data. The **WriteText** method does not truncate the rest of the data in a stream. If you want to truncate these characters, call [SetEOS](#).

If you write past the current [EOS](#) position, the [Size](#) of the **Stream** will be increased to contain any new characters, and **EOS** will move to the new last byte in the **Stream**.

Note The **WriteText** method is used with text streams ([Type](#) is **adTypeText**). For binary streams ([Type](#) is **adTypeBinary**), use [Write](#).

See Also

[Write Method](#)

Applies To: [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Events

<u>BeginTransComplete</u>	Called after the BeginTrans operation.
<u>CommitTransComplete</u>	Called after the CommitTrans operation.
<u>ConnectComplete</u>	Called after a connection starts.
<u>Disconnect</u>	Called after a connection ends.
<u>EndOfRecordset</u>	Called when there is an attempt to move to a row past the end of the Recordset .
<u>ExecuteComplete</u>	Called after a command has finished executing.
<u>FetchComplete</u>	Called after all the records in a lengthy asynchronous operation have been retrieved into the Recordset .
<u>FetchProgress</u>	Called periodically during a lengthy asynchronous operation to report how many rows have currently been retrieved into the Recordset .
<u>FieldChangeComplete</u>	Called after the value of one or more Field objects has changed.
<u>InfoMessage</u>	Called whenever a warning occurs during a ConnectionEvent operation.
<u>MoveComplete</u>	Called after the current position in the Recordset changes.
<u>RecordChangeComplete</u>	Called after one or more records change.
<u>RecordsetChangeComplete</u>	Called after the Recordset has changed.
<u>RollbackTransComplete</u>	Called after the RollbackTrans operation.
<u>WillChangeField</u>	Called before a pending operation changes the value of one or more Field objects in the Recordset .
<u>WillChangeRecord</u>	Called before one or more records (rows) in the Recordset change.
<u>WillChangeRecordset</u>	Called before a pending operation changes the Recordset .
<u>WillConnect</u>	Called before a connection starts. Called just before a pending command executes on this connection and affords the user an opportunity

[WillExecute](#)

to examine and modify the pending execution parameters.

[WillMove](#)

The **WillMove** event is called *before* a pending operation changes the current position in the **Recordset**.

See Also

[ADO API Reference](#) | [ADO Collections](#) | [ADO Dynamic Properties](#) | [ADO Enumerated Constants](#) | [ADO Errors](#) | [ADO Event Model, Synchronous and Asynchronous Operations](#) | [ADO Methods](#) | [ADO Object Model](#) | [ADO Objects](#) | [ADO Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

BeginTransComplete, CommitTransComplete, and RollbackTransComplete Events

These events will be called after the associated operation on the [Connection](#) object finishes executing.

- **BeginTransComplete** is called after the [BeginTrans](#) operation.
- **CommitTransComplete** is called after the [CommitTrans](#) operation.
- **RollbackTransComplete** is called after the [RollbackTrans](#) operation.

Syntax

```
BeginTransComplete TransactionLevel, pError, adStatus, pConnection  
CommitTransComplete pError, adStatus, pConnection  
RollbackTransComplete pError, adStatus, pConnection
```

Parameters

TransactionLevel

A **Long** value that contains the new transaction level of the **BeginTrans** that caused this event.

pError

An [Error](#) object. It describes the error that occurred if the value of `EventStatusEnum` is **adStatusErrorsOccurred**; otherwise it is not set.

adStatus

An [EventStatusEnum](#) status value. When any of these events is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful, or to **adStatusErrorsOccurred** if the operation failed.

These events can prevent subsequent notifications by setting this parameter to **adStatusUnwantedEvent** before the event returns.

pConnection

The **Connection** object for which this event occurred.

Remarks

In Visual C++, multiple **Connections** can share the same event handling method. The method uses the returned **Connection** object to determine which object caused the event.

If the [Attributes](#) property is set to **adXactCommitRetaining** or **adXactAbortRetaining**, a new transaction starts after committing or rolling back a transaction. Use the **BeginTransComplete** event to ignore all but the first transaction start event.

See Also

[Visual C++ Example](#) | [Visual J++ Example](#) | [Visual Basic Example](#)

[ADO Event Handler Summary](#) | [BeginTrans, CommitTrans, and RollbackTrans Methods](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ConnectComplete and Disconnect Events

The **ConnectComplete** event is called after a connection *starts*. The **Disconnect** event is called after a connection *ends*.

Syntax

ConnectComplete *pError, adStatus, pConnection*
Disconnect *adStatus, pConnection*

Parameters

pError

An [Error](#) object. It describes the error that occurred if the value of *adStatus* is **adStatusErrorsOccurred**; otherwise it is not set.

adStatus

An [EventStatusEnum](#) value that always returns **adStatusOK**.

When **ConnectComplete** is called, this parameter is set to **adStatusCancel** if a **WillConnect** event has requested cancellation of the pending connection.

Before either event returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications. However, closing and reopening the [Connection](#) causes these events to occur again.

pConnection

The **Connection** object for which this event applies.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

EndOfRecordset Event

The **EndOfRecordset** event is called when there is an attempt to move to a row past the end of the [Recordset](#).

Syntax

```
EndOfRecordset fMoreData, adStatus, pRecordset
```

Parameters

fMoreData

A **VARIANT_BOOL** value that, if set to **VARIANT_TRUE**, indicates more rows have been added to the **Recordset**.

adStatus

An [EventStatusEnum](#) status value.

When **EndOfRecordset** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful. It is set to **adStatusCantDeny** if this event cannot request cancellation of the operation that caused this event.

Before **EndOfRecordset** returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

pRecordset

A **Recordset** object. The **Recordset** for which this event occurred.

Remarks

An **EndOfRecordset** event may occur if the [MoveNext](#) operation fails.

This event handler is called when an attempt is made to move past the end of the **Recordset** object, perhaps as a result of calling **MoveNext**. However, while in this event, you could retrieve more records from a database and append them to the end of the **Recordset**. In that case, set *fMoreData* to **VARIANT_TRUE**, and

return from **EndOfRecordset**. Then call **MoveNext** again to access the newly retrieved records.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ExecuteComplete Event

The **ExecuteComplete** event is called after a command has finished executing.

Syntax

ExecuteComplete *RecordsAffected, pError, adStatus, pCommand, pRecordset*

Parameters

RecordsAffected

A **Long** value indicating the number of records affected by the command.

pError

An [Error](#) object. It describes the error that occurred if the value of **adStatus** is **adStatusErrorsOccurred**; otherwise it is not set.

adStatus

An [EventStatusEnum](#) status value. When this event is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful, or to **adStatusErrorsOccurred** if the operation failed.

Before this event returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

pCommand

The [Command](#) object that was executed. Contains a **Command** object even when calling **Connection.Execute** or **Recordset.Open** without explicitly creating a **Command**, in which cases the **Command** object is created internally by ADO.

pRecordset

A [Recordset](#) object that is the result of the executed command. This **Recordset** may be empty. You should never destroy this Recordset object from within this event handler. Doing so will result in an Access Violation when ADO tries to access an object that no longer exists.

pConnection

A [Connection](#) object. The connection over which the operation was executed.

Remarks

An **ExecuteComplete** event may occur due to the **Connection**.[Execute](#), **Command**.[Execute](#), **Recordset**.[Open](#), **Recordset**.[Requery](#), or **Recordset**.[NextRecordset](#) methods.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

FetchComplete Event

The **FetchComplete** event is called after all the records in a lengthy asynchronous operation have been retrieved into the [Recordset](#).

Syntax

FetchComplete *pError*, *adStatus*, *pRecordset*

Parameters

pError

An [Error](#) object. It describes the error that occurred if the value of **adStatus** is **adStatusErrorsOccurred**; otherwise it is not set.

adStatus

An [EventStatusEnum](#) status value. When this event is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful, or to **adStatusErrorsOccurred** if the operation failed.

Before this event returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

pRecordset

A **Recordset** object. The object for which the records were retrieved.

Remarks

To use **FetchComplete** with Microsoft Visual Basic, Visual Basic 6.0 or later is required.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

FetchProgress Event

The **FetchProgress** event is called periodically during a lengthy asynchronous operation to report how many more rows have currently been retrieved into the [Recordset](#).

Syntax

FetchProgress *Progress, MaxProgress, adStatus, pRecordset*

Parameters

Progress

A **Long** value indicating the number of records that have currently been retrieved by the fetch operation.

MaxProgress

A **Long** value indicating the maximum number of records expected to be retrieved.

adStatus

An [EventStatusEnum](#) status value.

pRecordset

A **Recordset** object that is the object for which the records are being retrieved.

Remarks

When using **FetchProgress** with a child **Recordset**, be aware that the *Progress* and *MaxProgress* parameter values are derived from the underlying [Cursor Service rowset](#). The values returned represent the total number of records in the underlying rowset, not just the number of records in the current [chapter](#).

Note To use **FetchProgress** with Microsoft Visual Basic, Visual Basic 6.0 or later is required.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

InfoMessage Event

The **InfoMessage** event is called whenever a warning occurs during a **ConnectionEvent** operation.

Syntax

InfoMessage *pError*, *adStatus*, *pConnection*

Parameters

pError

An [Error](#) object. This parameter contains any errors that are returned. If multiple errors are returned, enumerate the **Errors** collection to find them.

adStatus

An [EventStatusEnum](#) status value. If a warning occurs, *adStatus* is set to **adStatusOK** and the *pError* contains the warning.

Before this event returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

pConnection

A [Connection](#) object. The connection for which the warning occurred. For example, warnings can occur when opening a **Connection** object or executing a [Command](#) on a **Connection**.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Connection Object](#)

ADO 2.5 API Reference

WillChangeField and FieldChangeComplete Events

The **WillChangeField** event is called before a pending operation changes the value of one or more [Field](#) objects in the [Recordset](#). The **FieldChangeComplete** event is called after the value of one or more **Field** objects has changed.

Syntax

```
WillChangeField cFields, Fields, adStatus, pRecordset  
FieldChangeComplete cFields, Fields, pError, adStatus, pRecordset
```

Parameters

cFields

A **Long** that indicates the number of **Field** objects in *Fields*.

Fields

For **WillChangeField**, the *Fields* parameter is an array of **Variants** that contains **Field** objects with the original values.

For **FieldChangeComplete**, the *Fields* parameter is an array of **Variants** that contains **Field** objects with the changed values.

pError

An [Error](#) object. It describes the error that occurred if the value of *adStatus* is **adStatusErrorsOccurred**; otherwise it is not set.

adStatus

An [EventStatusEnum](#) status value.

When **WillChangeField** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful. It is set to **adStatusCantDeny** if this event cannot request cancellation of the pending operation.

When **FieldChangeComplete** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful, or to

adStatusErrorsOccurred if the operation failed.

Before **WillChangeField** returns, set this parameter to **adStatusCancel** to request cancellation of the pending operation.

Before **FieldChangeComplete** returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

pRecordset

A **Recordset** object. The **Recordset** for which this event occurred.

Remarks

A **WillChangeField** or **FieldChangeComplete** event may occur when setting the [Value](#) property and calling the [Update](#) method with field and value array parameters.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

WillChangeRecord and RecordChangeComplete Events

The **WillChangeRecord** event is called before one or more records (rows) in the [Recordset](#) change. The **RecordChangeComplete** event is called after one or more records change.

Syntax

```
WillChangeRecord adReason, cRecords, adStatus, pRecordset  
RecordChangeComplete adReason, cRecords, pError, adStatus, pRecordse
```

Parameters

adReason

An [EventReasonEnum](#) value that specifies the reason for this event. Its value can be **adRsnAddNew**, **adRsnDelete**, **adRsnUpdate**, **adRsnUndoUpdate**, **adRsnUndoAddNew**, **adRsnUndoDelete**, or **adRsnFirstChange**.

cRecords

A **Long** value that indicates the number of records changing (affected).

pError

An [Error](#) object. It describes the error that occurred if the value of *adStatus* is **adStatusErrorsOccurred**; otherwise it is not set.

adStatus

An [EventStatusEnum](#) status value.

When **WillChangeRecord** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful. It is set to **adStatusCantDeny** if this event cannot request cancellation of the pending operation.

When **RecordChangeComplete** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful, or to **adStatusErrorsOccurred** if the operation failed.

Before **WillChangeRecord** returns, set this parameter to **adStatusCancel** to request cancellation of the operation that caused this event or set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

Before **RecordChangeComplete** returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

pRecordset

A **Recordset** object. The **Recordset** for which this event occurred.

Remarks

A **WillChangeRecord** or **RecordChangeComplete** event may occur for the first changed field in a row due to the following **Recordset** operations: [Update](#), [Delete](#), [CancelUpdate](#), [AddNew](#), [UpdateBatch](#), and [CancelBatch](#). The value of the **Recordset** [CursorType](#) determines which operations cause the events to occur.

During the **WillChangeRecord** event, the **Recordset** [Filter](#) property is set to **adFilterAffectedRecords**. You cannot change this property while processing the event.

You must set the **adStatus** parameter to **adStatusUnwantedEvent** for each possible **adReason** value in order to completely stop event notification for any event that includes an **adReason** parameter.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

WillChangeRecordset and RecordsetChangeComplete Events

The **WillChangeRecordset** event is called before a pending operation changes the [Recordset](#). The **RecordsetChangeComplete** event is called after the **Recordset** has changed.

Syntax

```
WillChangeRecordset    adReason, adStatus, pRecordset  
RecordsetChangeComplete  adReason, pError, adStatus, pRecordset
```

Parameters

adReason

An [EventReasonEnum](#) value that specifies the reason for this event. Its value can be **adRsnRequery**, **adRsnResynch**, **adRsnClose**, **adRsnOpen**.

adStatus

An [EventStatusEnum](#) status value.

When **WillChangeRecordset** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful. It is set to **adStatusCantDeny** if this event cannot request cancellation of the pending operation.

When **RecordsetChangeComplete** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful, **adStatusErrorsOccurred** if the operation failed, or **adStatusCancel** if the operation associated with the previously accepted **WillChangeRecordset** event has been canceled.

Before **WillChangeRecordset** returns, set this parameter to **adStatusCancel** to request cancellation of the pending operation or set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

Before **WillChangeRecordset** or **RecordsetChangeComplete** returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

pError

An [Error](#) object. It describes the error that occurred if the value of *adStatus* is **adStatusErrorsOccurred**; otherwise it is not set.

pRecordset

A **Recordset** object. The **Recordset** for which this event occurred.

Remarks

A **WillChangeRecordset** or **RecordsetChangeComplete** event may occur due to the **Recordset** [Requery](#) or [Open](#) methods.

If the provider does not support bookmarks, a **RecordsetChange** event notification occurs each time new rows are retrieved from the provider. The frequency of this event depends on the **RecordsetCacheSize** property.

You must set the **adStatus** parameter to **adStatusUnwantedEvent** for each possible **adReason** value in order to completely stop event notification for any event that includes an **adReason** parameter.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

WillConnect Event

The **WillConnect** event is called before a connection starts.

Syntax

WillConnect *ConnectionString, UserID, Password, Options, adStatus, p*

Parameters

ConnectionString

A **String** that contains connection information for the pending connection.

UserID

A **String** that contains a user name for the pending connection.

Password

A **String** that contains a password for the pending connection.

Options

A **Long** value that indicates how the provider should evaluate the *ConnectionString*. Your only option is **adAsyncOpen**.

adStatus

An [EventStatusEnum](#) status value.

When this event is called, this parameter is set to **adStatusOK** by default. It is set to **adStatusCantDeny** if the event cannot request cancellation of the pending operation.

Before this event returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications. Set this parameter to **adStatusCancel** to request the connection operation that caused cancellation of this notification.

pConnection

The [Connection](#) object for which this event notification applies. Changes to the parameters of the **Connection** by the **WillConnect** event handler will have no effect on the **Connection**.

Remarks

When **WillConnect** is called, the *ConnectionString*, *UserID*, *Password*, and *Options* parameters are set to the values established by the operation that caused this event (the pending connection), and can be changed before the event returns. **WillConnect** may return a request that the pending connection be canceled.

When this event is canceled, **ConnectComplete** will be called with its *adStatus* parameter set to **adStatusErrorsOccurred**.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

WillExecute Event

The **WillExecute** event is called just before a pending command executes on a connection.

Syntax

WillExecute *Source, CursorType, LockType, Options, adStatus, pCommand*

Parameters

Source

A **String** that contains an SQL command or a stored procedure name.

CursorType

A [CursorTypeEnum](#) that contains the type of cursor for the **Recordset** that will be opened. With this parameter, you can change the cursor to any type during a **Recordset Open** operation. *CursorType* will be ignored for any other operation.

LockType

A [LockTypeEnum](#) that contains the lock type for the **Recordset** that will be opened. With this parameter, you can change the lock to any type during a **Recordset Open** operation. *LockType* will be ignored for any other operation.

Options

A **Long** value that indicates options that can be used to execute the command or open the **Recordset**.

adStatus

An [EventStatusEnum](#) status value that may be **adStatusCantDeny** or **adStatusOK** when this event is called. If it is **adStatusCantDeny**, this event may not request cancellation of the pending operation.

Before this event returns, set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications, or **adStatusCancel** to request cancellation of the operation that caused this event.

pCommand

The [Command](#) object for which this event notification applies.
pRecordset

The [Recordset](#) object for which this event notification applies.
pConnection

The [Connection](#) object for which this event notification applies.

Remarks

A **WillExecute** event may occur due to a **Connection.Execute**, **Command.Execute**, or **Recordset.Open** method. The *pConnection* parameter should always contain a valid reference to a **Connection** object. If the event is due to **Connection.Execute**, the *pRecordset* and *pCommand* parameters are set to **Nothing**. If the event is due to **Recordset.Open**, the *pRecordset* parameter will reference the **Recordset** object and the *pCommand* parameter is set to **Nothing**. If the event is due to **Command.Execute**, the *pCommand* parameter will reference the **Command** object and the *pRecordset* parameter is set to **Nothing**.

WillExecute allows you to examine and modify the pending execution parameters. This event may return a request that the pending command be canceled.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Connection Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

WillMove and MoveComplete Events

The **WillMove** event is called before a pending operation changes the current position in the [Recordset](#). The **MoveComplete** event is called after the current position in the **Recordset** changes.

Syntax

```
WillMove adReason, adStatus, pRecordset  
MoveComplete adReason, pError, adStatus, pRecordset
```

Parameters

adReason

An [EventReasonEnum](#) value that specifies the reason for this event. Its value can be **adRsnMoveFirst**, **adRsnMoveLast**, **adRsnMoveNext**, **adRsnMovePrevious**, **adRsnMove**, or **adRsnRequery**.

pError

An [Error](#) object. It describes the error that occurred if the value of *adStatus* is **adStatusErrorsOccurred**; otherwise it is not set.

adStatus

An [EventStatusEnum](#) status value.

When **WillMove** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful. It is set to **adStatusCantDeny** if this event cannot request cancellation of the pending operation.

When **MoveComplete** is called, this parameter is set to **adStatusOK** if the operation that caused the event was successful, or to **adStatusErrorsOccurred** if the operation failed.

Before **WillMove** returns, set this parameter to **adStatusCancel** to request cancellation of the pending operation or set this parameter to **adStatusUnwantedEvent** to prevent subsequent notifications.

Before **MoveComplete** returns, set this parameter to

adStatusUnwantedEvent to prevent subsequent notifications.

pRecordset

A [Recordset](#) object. The **Recordset** for which this event occurred.

Remarks

A **WillMove** or **MoveComplete** event may occur due to the following **Recordset** operations: [Open](#), [Move](#), [MoveFirst](#), [MoveLast](#), [MoveNext](#), [MovePrevious](#), [AddNew](#), and [Requery](#). These events may occur because of the following properties: [Filter](#), [Index](#), [Bookmark](#), [AbsolutePage](#), and [AbsolutePosition](#). These events also occur if a child **Recordset** has **Recordset** events connected and the parent **Recordset** is moved.

You must set the **adStatus** parameter to **adStatusUnwantedEvent** for each possible adReason value in order to completely stop event notification for any event that includes an adReason parameter.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Enumerated Constants

To assist in debugging, the ADO enumerations list a value for each constant. However, this value is purely advisory, and may change from one release of ADO to another. Your code should only depend on the name, not the actual value, of each enumerated constant.

[ADCPROP_ASYNC_THREAD_PRIORITY_ENUM](#)

For an RDS **Recordset** object, specifies the execution priority of the asynchronous thread that retrieves data.

[ADCPROP_AUTO_RECALC_ENUM](#)

Specifies when the **MSDataShape** provider recalculates aggregate and calculated columns in a hierarchical **Recordset**.

[ADCPROP_UPDATECRITERIA_ENUM](#)

Specifies which fields can be used to detect conflicts during an optimistic update of a row of the data source with a **Recordset** object.

[ADCPROP_UPDATERESYNC_ENUM](#)

Specifies whether the **UpdateBatch** method is followed by an implicit **Resync** method operation and if so, the scope of that operation.

[AffectEnum](#)

Specifies which records are affected by an operation.

[BookmarkEnum](#)

Specifies a bookmark indicating where the operation should begin.

[CommandTypeEnum](#)

Specifies how a command argument should be

[CompareEnum](#)

interpreted.

Specifies the relative position of two records represented by their bookmarks.

[ConnectModeEnum](#)

Specifies the available permissions for modifying data in a **Connection**, opening a **Record**, or specifying values for the **Mode** property of the **Record** and **Stream** objects.

[ConnectOptionEnum](#)

Specifies whether the **Open** method of a **Connection** object should return after (synchronously) or before (asynchronously) the connection is established.

[ConnectPromptEnum](#)

Specifies whether a dialog box should be displayed to prompt for missing parameters when opening a connection to an ODBC data source.

[CopyRecordOptionsEnum](#)

Specifies the behavior of the **CopyRecord** method.

[CursorLocationEnum](#)

Specifies the location of the cursor engine.

[CursorOptionEnum](#)

Specifies what functionality the **Supports** method should test for.

[CursorTypeEnum](#)

Specifies the type of cursor used in a **Recordset** object.

[DataTypeEnum](#)

Specifies the data type of a **Field**, **Parameter**, or **Property**.

<u>EditModeEnum</u>	Specifies the editing status of a record.
<u>ErrorValueEnum</u>	Specifies the type of ADO run-time error.
<u>EventReasonEnum</u>	Specifies the reason that caused an event to occur.
<u>EventStatusEnum</u>	Specifies the current status of the execution of an event.
<u>ExecuteOptionEnum</u>	Specifies how a provider should execute a command.
<u>FieldEnum</u>	Specifies the special fields referenced in a Record object's Fields collection.
<u>FieldAttributeEnum</u>	Specifies one or more attributes of a Field object.
<u>FieldStatusEnum</u>	Specifies the status of a Field object.
<u>FilterGroupEnum</u>	Specifies the group of records to be filtered from a Recordset .
<u>GetRowsOptionEnum</u>	Specifies how many records to retrieve from a Recordset .
<u>IsolationLevelEnum</u>	Specifies the level of transaction isolation for a Connection object.
<u>LineSeparatorsEnum</u>	Specifies the character used as a line separator in text Stream objects.
<u>LockTypeEnum</u>	Specifies the type of lock placed on records during editing.
<u>MarshalOptionsEnum</u>	Specifies which records should be returned to the server.
	Specifies the behavior of

[MoveRecordOptionsEnum](#)

the **Record** object
MoveRecord method.

[ObjectStateEnum](#)

Specifies whether an object is open or closed, connecting to a data source, executing a command, or fetching data.

[ParameterAttributesEnum](#)

Specifies the attributes of a **Parameter** object.

[ParameterDirectionEnum](#)

Specifies whether the **Parameter** represents an input parameter, an output parameter, or both, or if the parameter is the return value from a stored procedure.

[PersistFormatEnum](#)

Specifies the format in which to save a **Recordset**.

[PositionEnum](#)

Specifies the current position of the record pointer within a **Recordset**.

[PropertyAttributesEnum](#)

Specifies the attributes of a **Property** object.

[RecordCreateOptionsEnum](#)

Specifies for the **Record** object **Open** method whether an existing **Record** should be opened, or a new **Record** should be created.

[RecordOpenOptionsEnum](#)

Specifies options for opening a **Record**. These values may be combined by using an OR operator.

[RecordStatusEnum](#)

Specifies the status of a record with regard to batch updates and other bulk operations.

Specifies the type of

[RecordTypeEnum](#)

Record object.

[ResyncEnum](#)

Specifies whether underlying values are overwritten by a call to **Resync**.

[SaveOptionsEnum](#)

Specifies whether a file should be created or overwritten when saving from a **Stream** object. The values can be combined with an AND operator.

[SchemaEnum](#)

Specifies the type of schema **Recordset** that the **OpenSchema** method retrieves. Specifies the direction of a record search within a **Recordset**.

[SearchDirectionEnum](#)

Specifies the direction of a record search within a **Recordset**. Specifies the type of **Seek** to execute.

[SeekEnum](#)

Specifies the type of **Seek** to execute. Specifies options for opening a **Stream** object. The values can be combined with an AND operator.

[StreamOpenOptionsEnum](#)

Specifies options for opening a **Stream** object. The values can be combined with an AND operator. Specifies whether the whole stream or the next line should be read from a **Stream** object.

Specifies whether the whole stream or the next line

[StreamReadEnum](#)

should be read from a **Stream** object. Specifies the type of data stored in a **Stream** object.

[StreamTypeEnum](#)

Specifies the type of data stored in a **Stream** object. Specifies whether a line separator is appended to the string written to a **Stream** object.

[StreamWriteEnum](#)

Specifies whether a line separator is appended to the string written to a **Stream** object. Specifies the format when retrieving a **Recordset** as a string.

[StringFormatEnum](#)

Specifies the format when retrieving a **Recordset** as a string. Specifies the transaction attributes of a **Connection** object.

[XactAttributeEnum](#)

Specifies the transaction attributes of a **Connection** object.

See Also

[ADO API Reference](#) | [ADO Collections](#) | [ADO Dynamic Properties](#) | [ADO Errors](#) | [ADO Events](#) | [ADO Methods](#) | [ADO Object Model](#) | [ADO Objects](#) | [ADO Properties](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

ADCPROP_ASYNC_THREAD_PRIORITY

For an RDS [Recordset](#) object, specifies the execution priority of the asynchronous thread that retrieves data.

Use these constants with the **Recordset "Background Thread Priority"** [dynamic property](#), which is referenced in the ADO-to-OLE DB Dynamic Property index and documented in the [Microsoft Cursor Service for OLE DB](#) documentation.

Constant	Value	Description
adPriorityAboveNormal	4	Sets priority between normal and highest.
adPriorityBelowNormal	2	Sets priority between lowest and normal.
adPriorityHighest	5	Sets priority to the highest possible.
AdPriorityLowest	1	Sets priority to the lowest possible.
adPriorityNormal	3	Sets priority to normal.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.AdcPropAsyncThreadPriority.ABOVENORMAL
AdoEnums.AdcPropAsyncThreadPriority.BELOWNORMAL
AdoEnums.AdcPropAsyncThreadPriority.HIGHEST
AdoEnums.AdcPropAsyncThreadPriority.LOWEST
AdoEnums.AdcPropAsyncThreadPriority.NORMAL

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADCPROP_AUTORECALC_ENUM

Specifies when the [MSDataShape](#) provider re-calculates aggregate and calculated columns in a [hierarchical Recordset](#).

These constants are only used with the [MSDataShape provider](#) and the **Recordset "Auto Recalc"** [dynamic property](#), which is referenced in the [ADO Dynamic Property Index](#) and documented in the [Microsoft Cursor Service for OLE DB](#) or [Microsoft Data Shaping Service for OLE DB](#) documentation.

Constant	Value	Description
adRecalcAlways	1	Default. Recalculates whenever the MSDataShape provider determines values that the calculated columns depend upon have changed.
adRecalcUpFront	0	Calculates only when initially building the hierarchical Recordset .

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADCPROP_UPDATECRITERIA_EN

Specifies which fields can be used to detect conflicts during an [optimistic update](#) of a row of the data source with a [Recordset](#) object.

Use these constants with the **Recordset** "Update Criteria" [dynamic property](#), which is referenced in the [ADO Dynamic Property Index](#) and documented in the [Microsoft Cursor Service for OLE DB](#) documentation.

Constant	Value	Description
adCriteriaAllCols	1	Detects conflicts if any column of the data source row has been changed.
adCriteriaKey	0	Detects conflicts if the key column of the data source row has been changed, which means that the row has been deleted.
adCriteriaTimeStamp	3	Detects conflicts if the timestamp of the data source row has been changed, which means the row has been accessed after the Recordset was obtained.
adCriteriaUpdCols	2	Detects conflicts if any of the columns of the data source row that correspond to updated fields of the Recordset have been changed.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.AdcPropUpdateCriteria.ALLCOLS

AdoEnums.AdcPropUpdateCriteria.KEY

AdoEnums.AdcPropUpdateCriteria.TIMESTAMP

AdoEnums.AdcPropUpdateCriteria.UPDCOLS

ADO 2.5 API Reference

ADCPROP_UPDATERESYNC_ENUM

Specifies whether the [UpdateBatch](#) method is followed by an implicit [Resync](#) method operation and if so, the scope of that operation.

Constant	Value	Description
adResyncAll	15	Invokes Resync with the combined value of all the other ADCPROP_UPDATERESYNC_ENUM members.
adResyncAutoIncrement	1	Default. Attempts to retrieve the new identity value for columns that are automatically incremented or generated by the data source, such as Microsoft Jet AutoNumber fields or Microsoft SQL Server Identity columns.
adResyncConflicts	2	Invokes Resync for all rows in which the update or delete operation failed because of a concurrency conflict.
adResyncInserts	8	Invokes Resync for all successfully inserted rows. However, AutoIncrement column values are not resynchronized. Instead, the contents of newly inserted rows are resynchronized based on the existing primary key value. If the primary key is an AutoIncrement value, Resync won't retrieve the contents of the intended row. For automatically incrementing AutoIncrement primary key values, call UpdateBatch with the combined value adResyncAutoIncrement + adResyncInserts .
adResyncNone	0	Does not invoke Resync . Invokes Resync for all successfully

adResyncUpdates 4 updated rows.

See Also

Applies To: [Update Resync Property—Dynamic \(ADO\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

AffectEnum

Specifies which records are affected by an operation.

Constant	Value	Description
		If there is not a Filter applied to the Recordset , affects all records.
		If the Filter property is set to a string criteria (such as "Author='Smith'"), then the operation affects visible records in the current chapter .
adAffectAll	3	If the Filter property is set to a member of the FilterGroupEnum or an array of Bookmarks , then the operation will affect all rows of the Recordset .
		Note adAffectAll is hidden in the Visual Basic Object Browser.
adAffectAllChapters	4	Affects all records in all sibling chapters of the Recordset , including those not visible via any Filter that is currently applied.
adAffectCurrent	1	Affects only the current record.
adAffectGroup	2	Affects only records that satisfy the current Filter property setting. You must set the Filter property to a FilterGroupEnum value or an array of Bookmarks to use this option.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.Affect.ALL

AdoEnums.Affect.ALLCHAPTERS

AdoEnums.Affect.CURRENT

AdoEnums.Affect.GROUP

See Also

Applies To: [CancelBatch Method](#) | [Delete Method \(ADO Recordset\)](#) | [Resync Method](#) | [UpdateBatch Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

BookmarkEnum

Specifies a bookmark indicating where the operation should begin.

Constant	Value	Description
adBookmarkCurrent	0	Starts at the current record.
adBookmarkFirst	1	Starts at the first record.
adBookmarkLast	2	Starts at the last record.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.Bookmark.CURRENT

AdoEnums.Bookmark.FIRST

AdoEnums.Bookmark.LAST

See Also

Applies To: [GetRows Method](#) | [Move Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CommandTypeEnum

Specifies how a command argument should be interpreted.

Constant	Value	Description
adCmdUnspecified	-1	Does not specify the command type argument.
adCmdText	1	Evaluates CommandText as a textual definition of a command or stored procedure call.
adCmdTable	2	Evaluates CommandText as a table name whose columns are all returned by an internally generated SQL query.
adCmdStoredProc	4	Evaluates CommandText as a stored procedure name.
adCmdUnknown	8	Default. Indicates that the type of command in the CommandText property is not known.
adCmdFile	256	Evaluates CommandText as the file name of a persistently stored Recordset . Used with Recordset.Open or Requery only.
adCmdTableDirect	512	Evaluates CommandText as a table name whose columns are all returned. Used with Recordset.Open or Requery only. To use the Seek method, the Recordset must be opened with adCmdTableDirect . This value cannot be combined with the ExecuteOptionEnum value adAsyncExecute .

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.CommandType.UNSPECIFIED

AdoEnums.CommandType.TEXT

AdoEnums.CommandType.TABLE

AdoEnums.CommandType.STOREDPROC

AdoEnums.CommandType.UNKNOWN

AdoEnums.CommandType.FILE

AdoEnums.CommandType.TABLEDIRECT

See Also

Applies To: [CommandType Property](#) | [Execute Method \(ADO Command\)](#) | [Execute Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#) | [Requery Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CompareEnum

Specifies the relative position of two records represented by their bookmarks.

Constant	Value	Description
adCompareEqual	1	Indicates that the bookmarks are equal.
adCompareGreaterThan	2	Indicates that the first bookmark is after the second.
adCompareLessThan	0	Indicates that the first bookmark is before the second.
adCompareNotComparable	4	Indicates that the bookmarks cannot be compared.
adCompareNotEqual	3	Indicates that the bookmarks are not equal and not ordered.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.Compare.EQUAL
AdoEnums.Compare.GREATERTHAN
AdoEnums.Compare.LESSTHAN
AdoEnums.Compare.NOTCOMPARABLE
AdoEnums.Compare.NOTEQUAL

See Also

Applies To: [CompareBookmarks Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ConnectModeEnum

Specifies the available permissions for modifying data in a [Connection](#), opening a [Record](#), or specifying values for the [Mode](#) property of the **Record** and [Stream](#) objects.

Constant	Value	Description
adModeRead	1	Indicates read-only permissions.
adModeReadWrite	3	Indicates read/write permissions.
adModeRecursive	0x400000	Used in conjunction with the other <i>*ShareDeny*</i> values (adModeShareDenyNone , adModeShareDenyWrite , or adModeShareDenyRead) to propagate sharing restrictions to all sub-records of the current Record . It has no affect if the Record does not have any children. A run-time error is generated if it is used with adModeShareDenyNone only. However, it can be used with adModeShareDenyNone when combined with other values. For example, you can use " adModeRead Or adModeShareDenyNone Or adModeRecursive ".
adModeShareDenyNone	16	Allows others to open a connection with any permissions. Neither read nor write access can be denied to others.
adModeShareDenyRead	4	Prevents others from opening a connection with read permissions.
adModeShareDenyWrite	8	Prevents others from opening a connection with write permissions.
		Prevents others from opening a

adModeShareExclusive	12	connection.
adModeUnknown	0	Default. Indicates that the permissions have not yet been set or cannot be determined.
adModeWrite	2	Indicates write-only permissions.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.ConnectMode.READ

AdoEnums.ConnectMode.READWRITE

(There is no equivalent of AdoEnums.ConnectMode.RECURSIVE)

AdoEnums.ConnectMode.SHAREDENYNONE

AdoEnums.ConnectMode.SHAREDENYREAD

AdoEnums.ConnectMode.SHAREDENYWRITE

AdoEnums.ConnectMode.SHAREEXCLUSIVE

AdoEnums.ConnectMode.UNKNOWN

AdoEnums.ConnectMode.WRITE

See Also

Applies To: [Mode Property](#) | [Open Method \(ADO Record\)](#) | [Open Method \(ADO Stream\)](#) | [Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ConnectOptionEnum

Specifies whether the [Open](#) method of a [Connection](#) object should return after ([synchronously](#)) or before ([asynchronously](#)) the connection is established.

Constant	Value	Description
adAsyncConnect	16	Opens the connection asynchronously. The ConnectComplete event may be used to determine when the connection is available.
adConnectUnspecified	-1	Default. Opens the connection synchronously.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.ConnectOption.ASYNCCONNECT

AdoEnums.ConnectOption.CONNECTUNSPECIFIED

See Also

Applies To: [Open Method \(ADO Connection\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ConnectPromptEnum

Specifies whether a dialog box should be displayed to prompt for missing parameters when opening a connection to a data source.

Constant	Value	Description
adPromptAlways	1	Prompts always.
adPromptComplete	2	Prompts if more information is required.
adPromptCompleteRequired	3	Prompts if more information is required but optional parameters are not allowed.
adPromptNever	4	Never prompts.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.ConnectPrompt.ALWAYS

AdoEnums.ConnectPrompt.COMPLETE

AdoEnums.ConnectPrompt.COMPLETEREQUIRED

AdoEnums.ConnectPrompt.NEVER

See Also

Applies To: [Prompt Property—Dynamic \(ADO\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CopyRecordOptionsEnum

Specifies the behavior of the [CopyRecord](#) method.

Constant	Value	Description
adCopyAllowEmulation	4	Indicates that the <i>Source</i> provider attempts to simulate the copy using download and upload operations if this method fails due to <i>Destination</i> being on a different server or is serviced by a different provider than <i>Source</i> . Note that differing provider capabilities may hamper performance or lose data.
adCopyNonRecursive	2	Copies the current directory, but none of its subdirectories, to the destination. The copy operation is not recursive.
adCopyOverWrite	1	Overwrites the file or directory if the <i>Destination</i> points to an existing file or directory.
adCopyUnspecified	-1	Default. Performs the default copy operation: The operation fails if the destination file or directory already exists, and the operation copies recursively.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [CopyRecord Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CursorLocationEnum

Specifies the location of the cursor service.

Constant	Value	Description
adUseClient	3	Uses client-side cursors supplied by a local cursor library. Local cursor services often will allow many features that driver-supplied cursors may not, so using this setting may provide an advantage with respect to features that will be enabled. For backward compatibility, the synonym adUseClientBatch is also supported.
adUseNone	1	Does not use cursor services. (This constant is obsolete and appears solely for the sake of backward compatibility.)
adUseServer	2	Default. Uses data-provider or driver-supplied cursors. These cursors are sometimes very flexible and allow for additional sensitivity to changes others make to the data source. However, some features of the Microsoft Cursor Service for OLE DB (such as disassociated Recordset objects) cannot be simulated with server-side cursors and these features will be unavailable with this setting.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.CursorLocation.CLIENT

AdoEnums.CursorLocation.NONE

AdoEnums.CursorLocation.SERVER

See Also

Applies To: [CursorLocation Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

CursorOptionEnum

Specifies what functionality the [Supports](#) method should test for.

Constant	Value	Description
adAddNew	0x1000400	Supports the AddNew method to add new records.
adApproxPosition	0x4000	Supports the AbsolutePosition and AbsolutePage properties.
adBookmark	0x2000	Supports the Bookmark property to gain access to specific records.
adDelete	0x1000800	Supports the Delete method to delete records.
adFind	0x80000	Supports the Find method to locate a row in a Recordset .
adHoldRecords	0x100	Retrieves more records or changes the next position without committing all pending changes.
adIndex	0x100000	Supports the Index property to name an index.
adMovePrevious	0x200	Supports the MoveFirst and MovePrevious methods, and Move or GetRows methods to move the current record position backward without requiring bookmarks.
adNotify	0x40000	Indicates that the underlying data provider supports notifications (which determines whether Recordset events are supported).
adResync	0x20000	Supports the Resync method to update the cursor with the data that is visible in the underlying database.

adSeek	0x200000	Supports the Seek method to locate a row in a Recordset .
adUpdate	0x1008000	Supports the Update method to modify existing data.
adUpdateBatch	0x10000	Supports batch updating (UpdateBatch and CancelBatch methods) to transmit groups of changes to the provider.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.CursorOption.ADDNEW
 AdoEnums.CursorOption.APPROXPOSITION
 AdoEnums.CursorOption.BOOKMARK
 AdoEnums.CursorOption.DELETE
 AdoEnums.CursorOption.FIND
 AdoEnums.CursorOption.HOLDRECORDS
 AdoEnums.CursorOption.INDEX
 AdoEnums.CursorOption.MOVEPREVIOUS
 AdoEnums.CursorOption.NOTIFY
 AdoEnums.CursorOption.RESYNC
 AdoEnums.CursorOption.SEEK
 AdoEnums.CursorOption.UPDATE
 AdoEnums.CursorOption.UPDATEBATCH

See Also

Applies To: [Supports Method](#)

ADO 2.5 API Reference

CursorTypeEnum

Specifies the type of cursor used in a [Recordset](#) object.

Constant	Value	Description
adOpenDynamic	2	Uses a dynamic cursor . Additions, changes, and deletions by other users are visible, and all types of movement through the Recordset are allowed, except for bookmarks, if the provider doesn't support them.
adOpenForwardOnly	0	Default. Uses a forward-only cursor. Identical to a static cursor, except that you can only scroll forward through records. This improves performance when you need to make only one pass through a Recordset .
adOpenKeyset	1	Uses a keyset cursor. Like a dynamic cursor, except that you can't see records that other users add, although records that other users delete are inaccessible from your Recordset . Data changes by other users are still visible.
adOpenStatic	3	Uses a static cursor. A static copy of a set of records that you can use to find data or generate reports. Additions, changes, or deletions by other users are not visible.
adOpenUnspecified	-1	Does not specify the type of cursor.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.CursorType.DYNAMIC

AdoEnums.CursorType.FORWARDONLY

AdoEnums.CursorType.KEYSET

AdoEnums.CursorType.STATIC

AdoEnums.CursorType.UNSPECIFIED

See Also

Applies To: [CursorType Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

DataTypeEnum

Specifies the data type of a [Field](#), [Parameter](#), or [Property](#). The corresponding OLE DB type indicator is shown in parentheses in the description column of the following table. For more information about OLE DB data types, see Chapter 13 and Appendix A of the *OLE DB Programmer's Reference*.

Constant	Value	Description
AdArray (Does not apply to ADOX.)	0x2000	A flag value, always combined with another data type constant, that indicates an array of that other data type.
adBigInt	20	Indicates an eight-byte signed integer (DBTYPE_I8).
adBinary	128	Indicates a binary value (DBTYPE_BYTES).
adBoolean	11	Indicates a boolean value (DBTYPE_BOOL).
adBSTR	8	Indicates a null-terminated character string (Unicode) (DBTYPE_BSTR).
adChapter	136	Indicates a four-byte chapter value that identifies rows in a child rowset (DBTYPE_HCHAPTER).
adChar	129	Indicates a string value (DBTYPE_STR).
adCurrency	6	Indicates a currency value (DBTYPE_CY). Currency is a fixed-point number with four digits to the right of the decimal point. It is stored in an eight-byte signed integer scaled by 10,000.
adDate	7	Indicates a date value (DBTYPE_DATE). A date is stored as a double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
adDBDate	133	Indicates a date value (yyyymmdd)

		(DBTYPE_DBDATE).
adDBTime	134	Indicates a time value (hhmmss) (DBTYPE_DBTIME).
adDBTimeStamp	135	Indicates a date/time stamp (yyyymmddhhmmss plus a fraction in billionths) (DBTYPE_DBTIMESTAMP).
adDecimal	14	Indicates an exact numeric value with a fixed precision and scale (DBTYPE_DECIMAL).
adDouble	5	Indicates a double-precision floating-point value (DBTYPE_R8).
adEmpty	0	Specifies no value (DBTYPE_EMPTY).
adError	10	Indicates a 32-bit error code (DBTYPE_ERROR).
adFileTime	64	Indicates a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (DBTYPE_FILETIME).
adGUID	72	Indicates a globally unique identifier (GUID) (DBTYPE_GUID).
		Indicates a pointer to an IDispatch interface on a COM object (DBTYPE_IDISPATCH).
adIDispatch	9	Note This data type is currently not supported by ADO. Usage may cause unpredictable results.
adInteger	3	Indicates a four-byte signed integer (DBTYPE_I4).
		Indicates a pointer to an IUnknown interface on a COM object (DBTYPE_IUNKNOWN).
adIUnknown	13	Note This data type is currently not supported by ADO. Usage may cause unpredictable results.

adLongVarBinary	205	Indicates a long binary value.
adLongVarChar	201	Indicates a long string value.
adLongVarWChar	203	Indicates a long null-terminated Unicode string value.
adNumeric	131	Indicates an exact numeric value with a fixed precision and scale (DBTYPE_NUMERIC).
adPropVariant	138	Indicates an Automation PROPVARIANT (DBTYPE_PROP_VARIANT).
adSingle	4	Indicates a single-precision floating-point value (DBTYPE_R4).
adSmallInt	2	Indicates a two-byte signed integer (DBTYPE_I2).
adTinyInt	16	Indicates a one-byte signed integer (DBTYPE_I1).
adUnsignedBigInt	21	Indicates an eight-byte unsigned integer (DBTYPE_UI8).
adUnsignedInt	19	Indicates a four-byte unsigned integer (DBTYPE_UI4).
adUnsignedSmallInt	18	Indicates a two-byte unsigned integer (DBTYPE_UI2).
adUnsignedTinyInt	17	Indicates a one-byte unsigned integer (DBTYPE_UI1).
adUserDefined	132	Indicates a user-defined variable (DBTYPE_UDT).
adVarBinary	204	Indicates a binary value (Parameter object only).
adVarChar	200	Indicates a string value. Indicates an Automation Variant (DBTYPE_VARIANT).
adVariant	12	Note This data type is currently not supported by ADO. Usage may cause unpredictable results. Indicates a numeric value (Parameter

adVarNumeric	139	object only).
adVarWChar	202	Indicates a null-terminated Unicode character string.
adWChar	130	Indicates a null-terminated Unicode character string (DBTYPE_WSTR).

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.DataType.ARRAY
 AdoEnums.DataType.BIGINT
 AdoEnums.DataType.BINARY
 AdoEnums.DataType.BOOLEAN
 AdoEnums.DataType.BSTR
 AdoEnums.DataType.CHAPTER
 AdoEnums.DataType.CHAR
 AdoEnums.DataType.CURRENCY
 AdoEnums.DataType.DATE
 AdoEnums.DataType.DBDATE
 AdoEnums.DataType.DBTIME
 AdoEnums.DataType.DBTIMESTAMP
 AdoEnums.DataType.DECIMAL
 AdoEnums.DataType.DOUBLE
 AdoEnums.DataType.EMPTY
 AdoEnums.DataType.ERROR
 AdoEnums.DataType.FILETIME
 AdoEnums.DataType.GUID
 AdoEnums.DataType.IDISPATCH
 AdoEnums.DataType.INTEGER
 AdoEnums.DataType.IUNKNOWN
 AdoEnums.DataType.LONGVARBINARY

AdoEnums.DataType.LONGVARCHAR
AdoEnums.DataType.LONGVARWCHAR
AdoEnums.DataType.NUMERIC
AdoEnums.DataType.PROPVARIANT
AdoEnums.DataType.SINGLE
AdoEnums.DataType.SMALLINT
AdoEnums.DataType.TINYINT
AdoEnums.DataType.UNSIGNEDBIGINT
AdoEnums.DataType.UNSIGNEDINT
AdoEnums.DataType.UNSIGNEDSMALLINT
AdoEnums.DataType.UNSIGNEDTINYINT
AdoEnums.DataType.USERDEFINED
AdoEnums.DataType.VARBINARY
AdoEnums.DataType.VARCHAR
AdoEnums.DataType.VARIANT
AdoEnums.DataType.VARNUMERIC
AdoEnums.DataType.VARWCHAR
AdoEnums.DataType.WCHAR

See Also

Applies To: [Append Method](#) | [CreateParameter Method](#) | [CreateRecordset Method \(RDS\)](#) | [Type Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

EditModeEnum

Specifies the editing status of a record.

Constant	Value	Description
adEditNone	0	Indicates that no editing operation is in progress.
adEditInProgress	1	Indicates that data in the current record has been modified but not saved.
adEditAdd	2	Indicates that the AddNew method has been called, and the current record in the copy buffer is a new record that has not been saved in the database.
adEditDelete	4	Indicates that the current record has been deleted.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.EditMode.NONE

AdoEnums.EditMode.INPROGRESS

AdoEnums.EditMode.ADD

AdoEnums.EditMode.DELETE

See Also

Applies To: [EditMode Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ErrorValueEnum

Specifies the type of ADO run-time error.

Three forms of the error number are listed:

- Positive decimal—the low two bytes of the full number in decimal format. This number is displayed in the default Visual Basic error message dialog box. For example, Run-time error '3707'.
- Negative decimal—The decimal translation of the full error number.
- Hexadecimal—The hexadecimal representation of the full error number. The Windows facility code is in the fourth digit. The facility code for ADO error numbers is A. For example: 0x800A0E7B.

Note OLE DB errors may be passed to your ADO application. Typically, these can be identified by a Windows facility code of 4. For example, 0x8004.... For more information about these numbers, see Chapter 16 of the *OLE DB Programmer's Reference*.

Constant	Value	Description
adErrBoundToCommand	3707 -2146824581 0x800A0E7B	Cannot change the ActiveConnection property of a Recordset object which has a Command object as its source.
adErrCannotComplete	3732 -2146824556 0x800A0E94	Server cannot complete the operation.
adErrCantChangeConnection	3748 -2146824540 0x800A0EA4	Connection was denied. New connection you requested has different characteristics than the one already in use.
adErrCantChangeProvider	3220 -2146825068	Supplied provider is different from the one

	0X800A0C94	already in use.
adErrCantConvertvalue	3724 -2146824564 0x800A0E8C	Data value cannot be converted for reasons other than sign mismatch or data overflow. For example, conversion would have truncated data.
adErrCantCreate	3725 -2146824563 0x800A0E8D	Data value cannot be set or retrieved because the field data type was unknown, or the provider had insufficient resources to perform the operation.
adErrCatalogNotSet	3747 -2146824541 0x800A0EA3	Operation requires a valid ParentCatalog .
adErrColumnNotOnThisRow	3726 -2146824562 0x800A0E8E	Record does not contain this field.
adErrDataConversion	3421 -2146824867 0x800A0D5D	Application uses a value of the wrong type for the current operation.
adErrDataOverflow	3721 -2146824567 0x800A0E89	Data value is too large to be represented by the field data type.
adErrDelResOutOfScope	3738 -2146824550 0x800A0E9A	URL of the object to be deleted is outside the scope of the current record.
adErrDenyNotSupported	3750 -2146824538 0x800A0EA6	Provider does not support sharing restrictions.
adErrDenyTypeNotSupported	3751 -2146824537 0x800A0EA7	Provider does not support the requested kind of sharing restriction.
	3251	Object or provider is not

adErrFeatureNotAvailable	-2146825037 0x800A0CB3	capable of performing requested operation.
adErrFieldsUpdateFailed	3749 -2146824539 0x800A0EA5	Fields update failed. For further information, examine the Status property of individual field objects.
adErrIllegalOperation	3219 -2146825069 0x800A0C93	Operation is not allowed in this context.
adErrIntegrityViolation	3719 -2146824569 0x800A0E87	Data value conflicts with the integrity constraints of the field.
adErrInTransaction	3246 -2146825042 0x800A0CAE	Connection object cannot be explicitly closed while in a transaction.
adErrInvalidArgument	3001 -2146825287 0x800A0BB9	Arguments are of the wrong type, are out of acceptable range, or are in conflict with one another.
adErrInvalidConnection	3709 -2146824579 0x800A0E7D	The connection cannot be used to perform this operation. It is either closed or invalid in this context.
adErrInvalidParamInfo	3708 -2146824580 0x800A0E7C	Parameter object is improperly defined. Inconsistent or incomplete information was provided.
adErrInvalidTransaction	3714 -2146824574 0x800A0E82	Coordinating transaction is invalid or has not started.
adErrInvalidURL	3729 -2146824559	URL contains invalid characters. Make sure the

	0x800A0E91	URL is typed correctly.
	3265	Item cannot be found in the collection
adErrItemNotFound	-2146825023	corresponding to the requested name or ordinal.
	0x800A0CC1	
	3021	Either BOF or EOF is True, or the current record has been deleted.
adErrNoCurrentRecord	-2146825267	Requested operation requires a current record.
	0x800A0BCD	
	3715	Operation cannot be performed while not executing.
adErrNotExecuting	-2146824573	
	0x800A0E83	
	3710	Operation cannot be performed while processing event.
adErrNotReentrant	-2146824578	
	0x800A0E7E	
	3704	Operation is not allowed when the object is closed.
adErrObjectClosed	-2146824584	
	0x800A0E78	
	3367	Object is already in collection. Cannot append.
adErrObjectInCollection	-2146824921	
	0x800A0D27	
	3420	Object is no longer valid.
adErrObjectNotSet	-2146824868	
	0x800A0D5C	
	3705	Operation is not allowed when the object is open.
adErrObjectOpen	-2146824583	
	0x800A0E79	
	3002	File could not be opened.
adErrOpeningFile	-2146825286	
	0x800A0BBA	
	3712	Operation has been cancelled by the user.
adErrOperationCancelled	-2146824576	
	0x800A0E80	

Operation cannot be

adErrOutOfSpace	3734 -2146824554 0x800A0E96	performed. Provider cannot obtain enough storage space.
adErrPermissionDenied	3720 -2146824568 0x800A0E88	Insufficient permission prevents writing to the field.
adErrProviderFailed	3000 -2146825288 0x800A0BB8	Provider failed to perform the requested operation.
adErrProviderNotFound	3706 -2146824582 0x800A0E7A	Provider cannot be found. It may not be properly installed.
adErrReadFile	3003 -2146825285 0x800A0BBB	File could not be read.
adErrResourceExists	3731 -2146824557 0x800A0E93	Copy operation cannot be performed. Object named by destination URL already exists. Specify adCopyOverwrite to replace the object.
adErrResourceLocked	3730 -2146824558 0x800A0E92	Object represented by the specified URL is locked by one or more other processes. Wait until the process has finished and attempt the operation again.
adErrResourceOutOfScope	3735 -2146824553 0x800A0E97	Source or destination URL is outside the scope of the current record.
adErrSchemaViolation	3722 -2146824566 0x800A0E8A	Data value conflicts with the data type or constraints of the field.
	3723	Conversion failed because the data value

adErrSignMismatch	-2146824565 0x800A0E8B	was signed and the field data type used by the provider was unsigned.
adErrStillConnecting	3713 -2146824575 0x800A0E81	Operation cannot be performed while connecting asynchronously.
adErrStillExecuting	3711 -2146824577 0x800A0E7F	Operation cannot be performed while executing asynchronously.
adErrTreePermissionDenied	3728 -2146824560 0x800A0E90	Permissions are insufficient to access tree or subtree.
adErrUnavailable	3736 -2146824552 0x800A0E98	Operation failed to complete and the status is unavailable. The field may be unavailable or the operation was not attempted.
adErrUnsafeOperation	3716 -2146824572 0x800A0E84	Safety settings on this computer prohibit accessing a data source on another domain.
adErrURLDoesNotExist	3727 -2146824561 0x800A0E8F	Either the source URL or the parent of the destination URL does not exist.
adErrURLNamedRowDoesNotExist	3737 -2146824551 0x800A0E99	Record named by this URL does not exist.
adErrVolumeNotFound	3733 -2146824555 0x800A0E95	Provider cannot locate the storage device indicated by the URL. Make sure the URL is typed correctly.

adErrWriteFile	3004 -2146825284 0x800A0BBC	Write to file failed.
adWrnSecurityDialog	3717 -2146824571 0x800A0E85	For internal use only. Don't use.
adWrnSecurityDialogHeader	3718 -2146824570 0x800A0E86	For internal use only. Don't use.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Only the following subsets of ADO/WFC equivalents are defined.

Constant

AdoEnums.ErrorValue.BOUNDTOCOMMAND
 AdoEnums.ErrorValue.DATACONVERSION
 AdoEnums.ErrorValue.FEATURENOTAVAILABLE
 AdoEnums.ErrorValue.ILLEGALOPERATION
 AdoEnums.ErrorValue.INTRANSACTION
 AdoEnums.ErrorValue.INVALIDARGUMENT
 AdoEnums.ErrorValue.INVALIDCONNECTION
 AdoEnums.ErrorValue.INVALIDPARAMINFO
 AdoEnums.ErrorValue.ITEMNOTFOUND
 AdoEnums.ErrorValue.NOCURRENTRECORD
 AdoEnums.ErrorValue.NOTEXECUTING
 AdoEnums.ErrorValue.NOTREentrant
 AdoEnums.ErrorValue.OBJECTCLOSED
 AdoEnums.ErrorValue.OBJECTINCOLLECTION
 AdoEnums.ErrorValue.OBJECTNOTSET
 AdoEnums.ErrorValue.OBJECTOPEN
 AdoEnums.ErrorValue.OPERATIONCANCELLED
 AdoEnums.ErrorValue.PROVIDERNOTFOUND

AdoEnums.ErrorValue.STILLCONNECTING
AdoEnums.ErrorValue.STILLEXECUTING
AdoEnums.ErrorValue.UNSAFEOPERATION

See Also

[ADO Error Codes](#)

Applies To: [Number Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

EventReasonEnum

Specifies the reason that caused an event to occur.

Constant	Value	Description
adRsnAddNew	1	An operation added a new record.
adRsnClose	9	An operation closed the Recordset .
adRsnDelete	2	An operation deleted a record.
adRsnFirstChange	11	An operation made the first change to a record.
adRsnMove	10	An operation moved the record pointer within the Recordset .
adRsnMoveFirst	12	An operation moved the record pointer to the first record in the Recordset .
adRsnMoveLast	15	An operation moved the record pointer to the last record in the Recordset .
adRsnMoveNext	13	An operation moved the record pointer to the next record in the Recordset .
adRsnMovePrevious	14	An operation moved the record pointer to the previous record in the Recordset .
adRsnRequery	7	An operation requeryed the Recordset .
adRsnResynch	8	An operation resynchronized the Recordset with the database.
adRsnUndoAddNew	5	An operation reversed the addition of a new record.
adRsnUndoDelete	6	An operation reversed the deletion of a record.
adRsnUndoUpdate	4	An operation reversed the update of a record.
adRsnUpdate	3	An operation updated an existing record.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.EventReason.ADDNEW
AdoEnums.EventReason.CLOSE
AdoEnums.EventReason.DELETE
AdoEnums.EventReason.FIRSTCHANGE
AdoEnums.EventReason.MOVE
AdoEnums.EventReason.MOVEFIRST
AdoEnums.EventReason.MOVELAST
AdoEnums.EventReason.MOVENEXT
AdoEnums.EventReason.MOVEPREVIOUS
AdoEnums.EventReason.REQUERY
AdoEnums.EventReason.RESYNCH
AdoEnums.EventReason.UNDOADDNEW
AdoEnums.EventReason.UNDODELETE
AdoEnums.EventReason.UNDOUPDATE
AdoEnums.EventReason.UPDATE

See Also

Applies To: [WillChangeRecord and RecordChangeComplete Events](#) | [WillChangeRecordset and RecordsetChangeComplete Events](#) | [WillMove and MoveComplete Events](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

EventStatusEnum

Specifies the current status of the execution of an event.

Constant	Value	Description
<code>adStatusCancel</code>	4	Requests cancellation of the operation that caused the event to occur.
<code>adStatusCantDeny</code>	3	Indicates that the operation cannot request cancellation of the pending operation.
<code>adStatusErrorsOccurred</code>	2	Indicates that the operation that caused the event failed due to an error or errors.
<code>adStatusOK</code>	1	Indicates that the operation that caused the event was successful.
<code>adStatusUnwantedEvent</code>	5	Prevents subsequent notifications before the event method has finished executing.

ADO/WFC Equivalent

Package: `com.ms.wfc.data`

Constant

`AdoEnums.EventStatus.CANCEL`
`AdoEnums.EventStatus.CANTDENY`
`AdoEnums.EventStatus.ERRORSOCCURRED`
`AdoEnums.EventStatus.OK`
`AdoEnums.EventStatus.UNWANTEDEVENT`

See Also

Applies To: [BeginTransComplete](#), [CommitTransComplete](#), and [RollbackTransComplete Events](#) | [ConnectComplete and Disconnect Events](#) | [EndOfRecordset Event](#) | [ExecuteComplete Event](#) | [FetchComplete Event](#) | [InfoMessage Event](#) | [WillChangeField and FieldChangeComplete Events](#) | [WillChangeRecord and RecordChangeComplete Events](#) | [WillChangeRecordset](#)

[and RecordsetChangeComplete Events](#) | [WillConnect Event](#) | [WillExecute Event](#)
| [WillMove and MoveComplete Events](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ExecuteOptionEnum

Specifies how a provider should execute a command.

Constant	Value	Description
		Indicates that the command should execute asynchronously .
adAsyncExecute	0x10	This value cannot be combined with the CommandTypeEnum value adCmdTableDirect .
adAsyncFetch	0x20	Indicates that the remaining rows after the initial quantity specified in the CacheSize property should be retrieved asynchronously. Indicates that the main thread never blocks while retrieving. If the requested row has not been retrieved, the current row automatically moves to the end of the file.
adAsyncFetchNonBlocking	0x40	If you open a Recordset from a Stream containing a persistently stored Recordset , adAsyncFetchNonBlocking will not have an effect; the operation will be synchronous and blocking. adAsynchFetchNonBlocking has no effect when the adCmdTableDirect option is used to open the Recordset .
		Indicates that the command text is a command or stored procedure that does not return rows (for example, a command that only inserts data). If any rows are

adExecuteNoRecords	0x80	retrieved, they are discarded and not returned. adExecuteNoRecords can only be passed as an optional parameter to the Command or Connection Execute method. Indicates that the results of a command execution should be returned as a stream.
adExecuteStream	0x400	adExecuteStream can only be passed as an optional parameter to the Command Execute method. Indicates that the CommandText is a command or stored procedure that returns a single row which should be returned as a Record object.
adExecuteRecord		
adOptionUnspecified	-1	Indicates that the command is unspecified.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.ExecuteOption.ASYNCEXECUTE

AdoEnums.ExecuteOption.ASYNCFETCH

AdoEnums.ExecuteOption.ASYNCFETCHNONBLOCKING

AdoEnums.ExecuteOption.NO RECORDS

AdoEnums.ExecuteOption.UNSPECIFIED

See Also

Applies To: [Execute Method \(ADO Command\)](#) | [Execute Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#) | [Requery Method](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

FieldEnum

Specifies the special fields referenced in a [Record](#) object's [Fields](#) collection.

Remarks

These constants provide a "shortcut" to accessing special fields associated with a **Record**. Retrieve the [Field](#) object from the **Fields** collection, and then obtain its contents with the **Field** object's [Value](#) property.

Constant	Value	Description
adDefaultStream	-1	References the field containing the default Stream object associated with a Record .
adRecordURL	-2	References the field containing the absolute URL string for the current Record .

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

FieldAttributeEnum

Specifies one or more attributes of a [Field](#) object.

Constant	Value	Description
adFldCacheDeferred	0x1000	Indicates that the provider caches field values and that subsequent reads are done from the cache.
adFldFixed	0x10	Indicates that the field contains fixed-length data.
adFldIsChapter	0x2000	Indicates that the field contains a chapter value, which specifies a specific child recordset related to this parent field. Typically chapter fields are used with data shaping or filters.
adFldIsCollection	0x40000	Indicates that the field specifies that the resource represented by the record is a collection of other resources, such as a folder, rather than a simple resource, such as a text file.
adFldIsDefaultStream	0x20000	Indicates that the field contains the default stream for the resource represented by the record. For example, the default stream can be the HTML content of a root folder on a Web site, which is automatically served when the root URL is specified.
adFldIsNullable	0x20	Indicates that the field accepts null values.
adFldIsRowURL	0x10000	Indicates that the field contains the URL that names the resource from the data store represented by the record.

adFldLong	0x80	Indicates that the field is a long binary field. Also indicates that you can use the AppendChunk and GetChunk methods.
adFldMayBeNull	0x40	Indicates that you can read null values from the field.
adFldMayDefer	0x2	Indicates that the field is deferred—that is, the field values are not retrieved from the data source with the whole record, but only when you explicitly access them.
adFldNegativeScale	0x4000	Indicates that the field represents a numeric value from a column that supports negative scale values. The scale is specified by the NumericScale property.
adFldRowID	0x100	Indicates that the field contains a persistent row identifier that cannot be written to and has no meaningful value except to identify the row (such as a record number, unique identifier, and so forth).
adFldRowVersion	0x200	Indicates that the field contains some kind of time or date stamp used to track updates.
adFldUnknownUpdatable	0x8	Indicates that the provider cannot determine if you can write to the field.
adFldUnspecified	-1 0xFFFFFFFF	Indicates that the provider does not specify the field attributes.
adFldUpdatable	0x4	Indicates that you can write to the field.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.FieldAttribute.CACHEDEFERRED

AdoEnums.FieldAttribute.FIXED

AdoEnums.FieldAttribute.ISNULLABLE

AdoEnums.FieldAttribute.LONG

AdoEnums.FieldAttribute.MAYBENULL

AdoEnums.FieldAttribute.MAYDEFER

AdoEnums.FieldAttribute.NEGATIVESCALE

AdoEnums.FieldAttribute.ROWID

AdoEnums.FieldAttribute.ROWVERSION

AdoEnums.FieldAttribute.UNKNOWNUPDATABLE

AdoEnums.FieldAttribute.UNSPECIFIED

AdoEnums.FieldAttribute.UPDATABLE

See Also

Applies To: [Append Method](#) | [Attributes Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

FieldStatusEnum

Specifies the status of a **Field** object.

The **adFieldPending*** values indicate the operation that caused the status to be set, and may be combined with other status values.

Constant	Value	Description
adFieldAlreadyExists	26	Indicates that the specified field already exists.
adFieldBadStatus	12	Indicates that an invalid status value was sent from ADO to the OLE DB provider. Possible causes include an OLE DB 1.0 or 1.1 provider, or an improper combination of Value and Status .
adFieldCannotComplete	20	Indicates that the server of the URL specified by Source could not complete the operation.
adFieldCannotDeleteSource	23	Indicates that during a move operation, a tree or subtree was moved to a new location, but the source could not be deleted.
adFieldCantConvertValue	2	Indicates that the field cannot be retrieved or stored without loss of data.
adFieldCantCreate	7	Indicates that the field could not be added because the provider exceeded a limitation (such as the number of fields allowed).
		Indicates that the data returned

adFieldDataOverflow	6	from the provider overflowed the data type of the field.
adFieldDefault	13	Indicates that the default value for the field was used when setting data.
adFieldDoesNotExist	16	Indicates that the field specified does not exist.
adFieldIgnore	15	Indicates that this field was skipped when setting data values in the source. The provider set no value.
adFieldIntegrityViolation	10	Indicates that the field cannot be modified because it is a calculated or derived entity.
adFieldInvalidURL	17	Indicates that the data source URL contains invalid characters.
adFieldIsNull	3	Indicates that the provider returned a VARIANT value of type VT_NULL and that the field is not empty.
adFieldOK	0	Default. Indicates that the field was successfully added or deleted.
adFieldOutOfSpace	22	Indicates that the provider is unable to obtain enough storage space to complete a move or copy operation.
adFieldPendingChange	0x40000	Indicates either that the field has been deleted and then re-added, perhaps with a different data type, or that the value of the field that previously had a status of adFieldOK has changed. The final form of the field will modify the Fields collection after the Update

		method is called.
adFieldPendingDelete	0x20000	Indicates that the Delete operation caused the status to be set. The field has been marked for deletion from the Fields collection after the Update method is called.
adFieldPendingInsert	0x10000	Indicates that the Append operation caused the status to be set. The Field has been marked to be added to the Fields collection after the Update method is called.
adFieldPendingUnknown	0x80000	Indicates that the provider cannot determine what operation caused field status to be set.
adFieldPendingUnknownDelete	0x100000	Indicates that the provider cannot determine what operation caused field status to be set, and that the field will be deleted from the Fields collection after the Update method is called.
adFieldPermissionDenied	9	Indicates that the field cannot be modified because it is defined as read-only.
adFieldReadOnly	24	Indicates that the field in the data source is defined as read-only.
adFieldResourceExists	19	Indicates that the provider was unable to perform the operation because an object already exists at the destination URL and it is not able to overwrite the object.
		Indicates that the provider was

adFieldResourceLocked	18	unable to perform the operation because the data source is locked by one or more other application or process.
adFieldResourceOutOfScope	25	Indicates that a source or destination URL is outside the scope of the current record.
adFieldSchemaViolation	11	Indicates that the value violated the data source schema constraint for the field.
adFieldSignMismatch	5	Indicates that data value returned by the provider was signed but the data type of the ADO field value was unsigned.
adFieldTruncated	4	Indicates that variable-length data was truncated when reading from the data source.
adFieldUnavailable	8	Indicates that the provider could not determine the value when reading from the data source. For example, the row was just created, the default value for the column was not available, and a new value had not yet been specified.
adFieldVolumeNotFound	21	Indicates that the provider is unable to locate the storage volume indicated by the URL.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [Status Property \(ADO Field\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

FilterGroupEnum

Specifies the group of records to be filtered from a [Recordset](#).

Constant	Value	Description
adFilterAffectedRecords	2	Filters for viewing only records affected by the last Delete , Resync , UpdateBatch , or CancelBatch call.
adFilterConflictingRecords	5	Filters for viewing the records that failed the last batch update.
adFilterFetchedRecords	3	Filters for viewing the records in the current cache—that is, the results of the last call to retrieve records from the database.
adFilterNone	0	Removes the current filter and restores all records for viewing.
adFilterPendingRecords	1	Filters for viewing only records that have changed but have not yet been sent to the server. Applicable only for batch update mode.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.FilterGroup.AFFECTEDRECORDS

AdoEnums.FilterGroup.CONFLICTINGRECORDS

AdoEnums.FilterGroup.FETCHEDRECORDS

AdoEnums.FilterGroup.NONE

AdoEnums.FilterGroup.PENDINGRECORDS

See Also

Applies To: [Filter Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

GetRowsOptionEnum

Specifies how many records to retrieve from a [Recordset](#).

Constant	Value	Description
adGetRowsRest	-1	Retrieves the rest of the records in the Recordset , from either the current position or a bookmark specified by the <i>Start</i> parameter of the GetRows method.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.GetRowsOption.REST

See Also

Applies To: [GetRows Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

IsolationLevelEnum

Specifies the level of transaction isolation for a [Connection](#) object.

Constant	Value	Description
adXactUnspecified	-1	Indicates that the provider is using a different isolation level than specified, but that the level cannot be determined.
adXactChaos	16	Indicates that pending changes from more highly isolated transactions cannot be overwritten.
adXactBrowse	256	Indicates that from one transaction you can view uncommitted changes in other transactions.
adXactReadUncommitted	256	Same as adXactBrowse .
adXactCursorStability	4096	Indicates that from one transaction you can view changes in other transactions only after they have been committed.
adXactReadCommitted	4096	Same as adXactCursorStability .
adXactRepeatableRead	65536	Indicates that from one transaction you cannot see changes made in other transactions, but that requerying can retrieve new Recordset objects.
adXactIsolated	1048576	Indicates that transactions are conducted in isolation of other transactions.
adXactSerializable	1048576	Same as adXactIsolated .

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.IsolationLevel.UNSPECIFIED

AdoEnums.IsolationLevel.CHAOS

AdoEnums.IsolationLevel.BROWSE
AdoEnums.IsolationLevel.READUNCOMMITTED
AdoEnums.IsolationLevel.CURSORSTABILITY
AdoEnums.IsolationLevel.READCOMMITTED
AdoEnums.IsolationLevel.REPEATABLEREAD
AdoEnums.IsolationLevel.ISOLATED
AdoEnums.IsolationLevel.SERIALIZABLE

See Also

Applies To: [IsolationLevel Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

LineSeparatorsEnum

Specifies the character used as a line separator in text [Stream](#) objects.

Constant	Value	Description
adCR	13	Indicates carriage return.
adCRLF	-1	Default. Indicates carriage return line feed.
adLF	10	Indicates line feed.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [LineSeparator Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

LockTypeEnum

Specifies the type of lock placed on records during editing.

Constant	Value	Description
adLockBatchOptimistic	4	Indicates optimistic batch updates. Required for batch update mode.
adLockOptimistic	3	Indicates optimistic locking, record by record. The provider uses optimistic locking, locking records only when you call the Update method.
adLockPessimistic	2	Indicates pessimistic locking, record by record. The provider does what is necessary to ensure successful editing of the records, usually by locking records at the data source immediately after editing.
adLockReadOnly	1	Indicates read-only records. You cannot alter the data.
adLockUnspecified	-1	Does not specify a type of lock. For clones, the clone is created with the same lock type as the original.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.LockType.BATCHOPTIMISTIC

AdoEnums.LockType.OPTIMISTIC

AdoEnums.LockType.PESSIMISTIC

AdoEnums.LockType.READONLY

AdoEnums.LockType.UNSPECIFIED

See Also

Applies To: [Clone Method](#) | [LockType Property](#) | [Open Method \(ADO Recordset\)](#) | [WillExecute Event](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

MarshalOptionsEnum

Specifies which records should be returned to the server.

Constant	Value	Description
adMarshalAll	0	Default. Returns all rows to the server.
adMarshalModifiedOnly	1	Returns only modified rows to the server.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.MarshalOptions.ALL

AdoEnums.MarshalOptions.MODIFIEDONLY

See Also

Applies To: [MarshalOptions Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

MoveRecordOptionsEnum

Specifies the behavior of the [Record](#) object [MoveRecord](#) method.

Constant	Value	Description
adMoveUnspecified	-1	Default. Performs the default move operation: The operation fails if the destination file or directory already exists, and the operation updates hypertext links.
adMoveOverWrite	1	Overwrites the destination file or directory, even if it already exists.
adMoveDontUpdateLinks	2	Modifies the default behavior of MoveRecord method by not updating the hypertext links of the source Record . The default behavior depends on the capabilities of the provider. Move operation updates links if the provider is capable. If the provider cannot fix links or if this value is not specified, then the move succeeds even when links have not been fixed.
adMoveAllowEmulation	4	Requests that the provider attempt to simulate the move (using download, upload, and delete operations). If the attempt to move the Record fails because the destination URL is on a different server or serviced by a different provider than the source, this may cause increased latency or data loss, due to different provider capabilities when moving resources between providers.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [MoveRecord Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ObjectStateEnum

Specifies whether an object is open or closed, connecting to a data source, executing a command, or retrieving data.

Constant	Value	Description
adStateClosed	0	Indicates that the object is closed.
adStateOpen	1	Indicates that the object is open.
adStateConnecting	2	Indicates that the object is connecting.
adStateExecuting	4	Indicates that the object is executing a command.
adStateFetching	8	Indicates that the rows of the object are being retrieved.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.ObjectState.CLOSED

AdoEnums.ObjectState.OPEN

AdoEnums.ObjectState.CONNECTING

AdoEnums.ObjectState.EXECUTING

AdoEnums.ObjectState.FETCHING

See Also

Applies To: [State Property](#) | [State Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ParameterAttributesEnum

Specifies the attributes of a [Parameter](#) object.

Constant	Value	Description
adParamSigned	16	Indicates that the parameter accepts signed values.
adParamNullable	64	Indicates that the parameter accepts null values.
adParamLong	128	Indicates that the parameter accepts long binary data.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.ParameterAttributes.SIGNED

AdoEnums.ParameterAttributes.NULLABLE

AdoEnums.ParameterAttributes.LONG

See Also

Applies To: [Attributes Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ParameterDirectionEnum

Specifies whether the [Parameter](#) represents an input parameter, an output parameter, both an input and an output parameter, or the return value from a stored procedure.

Constant	Value	Description
adParamInput	1	Default. Indicates that the parameter represents an input parameter.
adParamInputOutput	3	Indicates that the parameter represents both an input and output parameter.
adParamOutput	2	Indicates that the parameter represents an output parameter.
adParamReturnValue	4	Indicates that the parameter represents a return value.
adParamUnknown	0	Indicates that the parameter direction is unknown.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.ParameterDirection.INPUT
AdoEnums.ParameterDirection.INPUTOUTPUT
AdoEnums.ParameterDirection.OUTPUT
AdoEnums.ParameterDirection.RETURNVALUE
AdoEnums.ParameterDirection.UNKNOWN

See Also

Applies To: [CreateParameter Method](#) | [Direction Property](#)

ADO 2.5 API Reference

PersistFormatEnum

Specifies the format in which to save a [Recordset](#).

Constant	Value	Description
adPersistADTG	0	Indicates Microsoft Advanced Data TableGram (ADTG) format.
adPersistADO	1	Indicates that ADO's own Extensible Markup Language (XML) format will be used. This value is the same as adPersistXML and is included for backwards compatibility.
adPersistXML	1	Indicates Extensible Markup Language (XML) format.
adPersistProviderSpecific	2	Indicates that the provider will persist the Recordset using its own format.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.PersistFormat.ADTG

AdoEnums.PersistFormat.XML

See Also

Applies To: [Save Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

PositionEnum

Specifies the current position of the record pointer within a [Recordset](#).

Constant	Value	Description
adPosBOF	-2	Indicates that the current record pointer is at BOF (that is, the BOF property is True).
adPosEOF	-3	Indicates that the current record pointer is at EOF (that is, the EOF property is True).
adPosUnknown	-1	Indicates that the Recordset is empty, the current position is unknown, or the provider does not support the AbsolutePage or AbsolutePosition property.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.Position.BOF

AdoEnums.Position.EOF

AdoEnums.Position.UNKNOWN

See Also

Applies To: [AbsolutePage Property](#) | [AbsolutePosition Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

PropertyAttributesEnum

Specifies the attributes of a [Property](#) object.

Constant	Value	Description
adPropNotSupported	0	Indicates that the property is not supported by the provider .
adPropRequired	1	Indicates that the user must specify a value for this property before the data source is initialized.
adPropOptional	2	Indicates that the user does not need to specify a value for this property before the data source is initialized.
adPropRead	512	Indicates that the user can read the property.
adPropWrite	1024	Indicates that the user can set the property.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.PropertyAttributes.NOTSUPPORTED

AdoEnums.PropertyAttributes.REQUIRED

AdoEnums.PropertyAttributes.OPTIONAL

AdoEnums.PropertyAttributes.READ

AdoEnums.PropertyAttributes.WRITE

See Also

Applies To: [Attributes Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

RecordCreateOptionsEnum

Specifies whether an existing **Record** should be opened or a new **Record** created for the [Record](#) object [Open](#) method. The values can be combined with an AND operator.

Constant	Value	Description
adCreateCollection	0x2000	Creates a new Record at the node specified by <i>Source</i> parameter, instead of opening an existing Record . If the source points to an existing node, then a run-time error occurs, unless adCreateCollection is combined with adOpenIfExists or adCreateOverwrite .
adCreateNonCollection	0	Creates a new Record of type adSimpleRecord .
adCreateOverwrite	0x4000000	Modifies the creation flags adCreateCollection , adCreateNonCollection , and adCreateStructDoc . When OR is used with this value and one of the creation flag values, if the source URL points to an existing node or Record , then the existing Record is overwritten and a new one is created in its place. This value cannot be used together with adOpenIfExists .
adCreateStructDoc	0x80000000	Creates a new Record of type adStructDoc , instead of opening an existing Record .
adFailIfNotExists	-1	Default. Results in a run-time error if <i>Source</i> points to a non-existent node. Modifies the creation flags adCreateCollection ,

adOpenIfExists

0x2000000

adCreateNonCollection, and **adCreateStructDoc**. When OR is used with this value and one of the creation flag values, if the source URL points to an existing node or **Record** object, then the [provider](#) must open the existing **Record** instead of creating a new one. This value cannot be used together with **adCreateOverwrite**.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [Open Method \(ADO Record\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

RecordOpenOptionsEnum

Specifies options for opening a [Record](#). These values may be combined by using OR.

Constant	Value	Description
adDelayFetchFields	0x8000	Indicates to the provider that the fields associated with the Record need not be retrieved initially, but can be retrieved at the first attempt to access the field. The default behavior, indicated by the absence of this flag, is to retrieve all the Record object fields.
adDelayFetchStream	0x4000	Indicates to the provider that the default stream associated with the Record need not be retrieved initially. The default behavior, indicated by the absence of this flag, is to retrieve the default stream associated with the Record object.
adOpenAsync	0x1000	Indicates that the Record object is opened in asynchronous mode.
adOpenExecuteCommand	0x10000	Indicates that the Source string contains command text that should be executed. This value is equivalent to the adCmdText option on Recordset.Open .
adOpenRecordUnspecified	-1	Default. Indicates no options are specified.
adOpenOutput	0x800000	Indicates that if the source points to a node that contains an executable script (such as an .ASP page), then the opened Record will contain the results of the executed script. This value is only valid with non-collection records.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [Open Method \(ADO Record\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

RecordStatusEnum

Specifies the status of a record with regard to batch updates and other bulk operations.

Constant	Value	Description
adRecCanceled	0x100	Indicates that the record was not saved because the operation was canceled.
adRecCantRelease	0x400	Indicates that the new record was not saved because the existing record was locked.
adRecConcurrencyViolation	0x800	Indicates that the record was not saved because optimistic concurrency was in use.
adRecDBDeleted	0x40000	Indicates that the record has already been deleted from the data source.
adRecDeleted	0x4	Indicates that the record was deleted.
adRecIntegrityViolation	0x1000	Indicates that the record was not saved because the user violated integrity constraints.
adRecInvalid	0x10	Indicates that the record was not saved because its bookmark is invalid.
adRecMaxChangesExceeded	0x2000	Indicates that the record was not saved because there were too many pending changes.
adRecModified	0x2	Indicates that the record was modified.
adRecMultipleChanges	0x40	Indicates that the record was not saved because it would have affected multiple records.
adRecNew	0x1	Indicates that the record is new.
adRecObjectOpen	0x4000	Indicates that the record was not saved because of a conflict with an

adRecOK	0	open storage object. Indicates that the record was successfully updated.
adRecOutOfMemory	0x8000	Indicates that the record was not saved because the computer has run out of memory.
adRecPendingChanges	0x80	Indicates that the record was not saved because it refers to a pending insert.
adRecPermissionDenied	0x10000	Indicates that the record was not saved because the user has insufficient permissions.
adRecSchemaViolation	0x20000	Indicates that the record was not saved because it violates the structure of the underlying database.
adRecUnmodified	0x8	Indicates that the record was not modified.

ADO/WFC Equivalent

AdoEnums.RecordStatus.

Package: **com.ms.wfc.data**

Constant

AdoEnums.RecordStatus.CANCELED
 AdoEnums.RecordStatus.CANTRELEASE
 AdoEnums.RecordStatus.CONCURRENCYVIOLATION
 AdoEnums.RecordStatus.DBDELETED
 AdoEnums.RecordStatus.DELETED
 AdoEnums.RecordStatus.INTEGRITYVIOLATION
 AdoEnums.RecordStatus.INVALID
 AdoEnums.RecordStatus.MAXCHANGESEXCEEDED
 AdoEnums.RecordStatus.MODIFIED
 AdoEnums.RecordStatus.MULTIPLECHANGES
 AdoEnums.RecordStatus.NEW

AdoEnums.RecordStatus.OBJECTOPEN
AdoEnums.RecordStatus.OK
AdoEnums.RecordStatus.OUTOFMEMORY
AdoEnums.RecordStatus.PENDINGCHANGES
AdoEnums.RecordStatus.PERMISSIONDENIED
AdoEnums.RecordStatus.SCHEMAVIOLATION
AdoEnums.RecordStatus.UNMODIFIED

See Also

Applies To: [Status Property \(ADO Recordset\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

RecordTypeEnum

Specifies the type of [Record](#) object.

Constant	Value	Description
adSimpleRecord	0	Indicates a <i>simple</i> record (does not contain child nodes).
adCollectionRecord	1	Indicates a <i>collection</i> record (contains child nodes).
adRecordUnknown	-1	Indicates that the type of this Record is unknown.
adStructDoc	2	Indicates a special kind of <i>collection</i> record that represents COM structured documents.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [RecordType Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ResyncEnum

Specifies whether underlying values are overwritten by a call to [Resync](#).

Constant	Value	Description
adResyncAllValues	2	Default. Overwrites data, and pending updates are canceled.
adResyncUnderlyingValues	1	Does not overwrite data, and pending updates are not canceled.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.Resync.ALLVALUES

AdoEnums.Resync.UNDERLYINGVALUES

See Also

Applies To: [Resync Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

SaveOptionsEnum

Specifies whether a file should be created or overwritten when saving from a [Stream](#) object. The values can be combined with an AND operator.

Constant	Value	Description
adSaveCreateNotExist	1	Default. Creates a new file if the file specified by the <i>FileName</i> parameter does not already exist.
adSaveCreateOverWrite	2	Overwrites the file with the data from the currently open Stream object, if the file specified by the <i>Filename</i> parameter already exists.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [SaveToFile Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

SchemaEnum

Specifies the type of schema **Recordset** that the [OpenSchema](#) method retrieves.

Remarks

Additional information about the function and columns returned for each ADO constant can be found in topics of Appendix B of the *OLE DB Programmers Reference*. The name of each topic is listed in parentheses in the Description section of the table below.

Additional information about the function and columns returned for each ADO MD constant can be found in topics of Chapter 23 of the *OLE DB for OLAP* documentation. The name of each topic is listed in parentheses and marked with an asterisk (*) in the Description column of the table below.

Translate the data types of columns in the OLE DB documentation to ADO data types by referring to the Description column of the ADO [DataTypeEnum](#) topic. For example, an OLE DB data type of **DBTYPE_WSTR** is equivalent to an ADO data type of **adWChar**.

ADO generates schema-like results for the constants, **adSchemaDBInfoKeywords** and **adSchemaDBInfoLiterals**. ADO creates a **Recordset**, then fills each row with the values returned respectively by the **IDBInfo::GetKeywords** and **IDBInfo::GetLiteralInfo** methods. Additional information about these methods can be found in the IDBInfo section of the *OLE DB Programmer's Reference*.

Constant	Value	Description
adSchemaAsserts	0	Returns the assertions defined in the catalog that are owned by a given user. (ASSERTIONS Rowset) Returns the physical attributes associated with catalogs accessible

adSchemaCatalogs	1	from the DBMS. (CATALOGS Rowset)
adSchemaCharacterSets	2	Returns the character sets defined in the catalog that are accessible to a given user. (CHARACTER_SETS Rowset)
adSchemaCheckConstraints	5	Returns the check constraints define in the catalog that are owned by a given user. (CHECK_CONSTRAINTS Rowset)
adSchemaCollations	3	Returns the character collations defined in the catalog that are accessible to a given user. (COLLATIONS Rowset)
adSchemaColumnPrivileges	13	Returns the privileges on columns o: tables defined in the catalog that are available to, or granted by, a given user. (COLUMN_PRIVILEGES Rowset)
adSchemaColumns	4	Returns the columns of tables (including views) defined in the catalog that are accessible to a given user. (COLUMNS Rowset)
		Returns the columns defined in the catalog that are dependent on a domain defined in the catalog and

adSchemaColumnsDomainUsage	11	owned by a given user. (COLUMN_DOMAIN_USAGE Rowset) Returns the columns used by referential constraints, unique constraints, check constraints, and assertions, defined in the catalog and owned by a given user.
adSchemaConstraintColumnUsage	6	(CONSTRAINT_COLUMN_USAC Rowset) Returns the tables that are used by referential constraints, unique constraints, check constraints, and assertions defined in the catalog and owned by a given user.
adSchemaConstraintTableUsage	7	(CONSTRAINT_TABLE_USAGE Rowset) Returns information about the available cubes in a schema (or the catalog, if the provider does not support schemas).
adSchemaCubes	32	(CUBES Rowset*) Returns a list of provider-specific keywords.
adSchemaDBInfoKeywords	30	(IDBInfo::GetKeywords *) Returns a list of provider-specific literals used in text commands.
adSchemaDBInfoLiterals	31	(IDBInfo::GetLiteralInfo *)

adSchemaDimensions	33	Returns information about the dimensions in a given cube. It has one row for each dimension. (DIMENSIONS Rowset *)
adSchemaForeignKeys	27	Returns the foreign key columns defined in the catalog by a given user. (FOREIGN_KEYS Rowset)
adSchemaHierarchies	34	Returns information about the hierarchies available in a dimension. (HIERARCHIES Rowset *)
adSchemaIndexes	12	Returns the indexes defined in the catalog that are owned by a given user. (INDEXES Rowset)
adSchemaKeyColumnUsage	8	Returns the columns defined in the catalog that are constrained as keys by a given user. (KEY_COLUMN_USAGE Rowset)
adSchemaLevels	35	Returns information about the levels available in a dimension. (LEVELS Rowset*)

adSchemaMeasures	36	Returns information about the available measures. (MEASURES Rowset *)
adSchemaMembers	38	Returns information about the available members. (MEMBERS Rowset *)
adSchemaPrimaryKeys	28	Returns the primary key columns defined in the catalog by a given use (PRIMARY_KEYS Rowset)
adSchemaProcedureColumns	29	Returns information about the columns of rowsets returned by procedures. (PROCEDURE_COLUMNS Rowse
adSchemaProcedureParameters	26	Returns information about the parameters and return codes of procedures. (PROCEDURE_PARAMETERS Rowset)
		Returns the procedures defined in th catalog that are owned by a given

adSchemaProcedures	16	user. (PROCEDURES Rowset)
adSchemaProperties	37	Returns information about the available properties for each level of the dimension. (PROPERTIES Rowset *)
adSchemaProviderSpecific	-1	Used if the provider defines its own nonstandard schema queries. Returns the (base) data types supported by the data provider.
adSchemaProviderTypes	22	(PROVIDER_TYPES Rowset)
AdSchemaReferentialConstraints	9	Returns the referential constraints defined in the catalog that are owned by a given user. (REFERENTIAL_CONSTRAINTS Rowset)
adSchemaSchemata	17	Returns the schemas (database objects) that are owned by a given user. (SCHEMATA Rowset)
adSchemaSQLLanguages	18	Returns the conformance levels, options, and dialects supported by the SQL-implementation processing dat defined in the catalog. (SQL_LANGUAGES Rowset)

adSchemaStatistics	19	Returns the statistics defined in the catalog that are owned by a given user. (STATISTICS Rowset)
adSchemaTableConstraints	10	Returns the table constraints defined in the catalog that are owned by a given user. (TABLE_CONSTRAINTS Rowset)
adSchemaTablePrivileges	14	Returns the privileges on tables defined in the catalog that are available to, or granted by, a given user. (TABLE_PRIVILEGES Rowset)
adSchemaTables	20	Returns the tables (including views) defined in the catalog that are accessible to a given user. (TABLES Rowset)
adSchemaTranslations	21	Returns the character translations defined in the catalog that are accessible to a given user. (TRANSLATIONS Rowset)
adSchemaTrustees	39	Reserved for future use.
adSchemaUsagePrivileges	15	Returns the USAGE privileges on objects defined in the catalog that are available to, or granted by, a given user.

		(USAGE_PRIVILEGES Rowset)
adSchemaViewColumnUsage	24	Returns the columns on which view tables, defined in the catalog and owned by a given user, are dependent. (VIEW_COLUMN_USAGE Rowset)
adSchemaViews	23	Returns the views defined in the catalog that are accessible to a given user. (VIEWS Rowset)
adSchemaViewTableUsage	25	Returns the tables on which viewed tables, defined in the catalog and owned by a given user, are dependent. (VIEW_TABLE_USAGE Rowset)

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.Schema.ASSERTS
 AdoEnums.Schema.CATALOGS
 AdoEnums.Schema.CHARACTERSETS
 AdoEnums.Schema.CHECKCONSTRAINTS
 AdoEnums.Schema.COLLATIONS
 AdoEnums.Schema.COLUMNPRIVILEGES
 AdoEnums.Schema.COLUMNS
 AdoEnums.Schema.COLUMNSDOMAINUSAGE
 AdoEnums.Schema.CONSTRAINTCOLUMNUSAGE
 AdoEnums.Schema.CONSTRAINTTABLEUSAGE
 AdoEnums.Schema.CUBES

AdoEnums.Schema.DBINFOKEYWORDS
AdoEnums.Schema.DBINFOLITERALS
AdoEnums.Schema.DIMENSIONS
AdoEnums.Schema.FOREIGNKEYS
AdoEnums.Schema.HIERARCHIES
AdoEnums.Schema.INDEXES
AdoEnums.Schema.KEYCOLUMNUSAGE
AdoEnums.Schema.LEVELS
AdoEnums.Schema.MEASURES
AdoEnums.Schema.MEMBERS
AdoEnums.Schema.PRIMARYKEYS
AdoEnums.Schema.PROCEDURECOLUMNS
AdoEnums.Schema.PROCEDUREPARAMETERS
AdoEnums.Schema.PROCEDURES
AdoEnums.Schema.PROPERTIES
AdoEnums.Schema.PROVIDERSPECIFIC
AdoEnums.Schema.PROVIDERTYPES
AdoEnums.Schema.REFERENTIALCONSTRAINTS
AdoEnums.Schema.SCHEMATA
AdoEnums.Schema.SQLLANGUAGES
AdoEnums.Schema.STATISTICS
AdoEnums.Schema.TABLECONSTRAINTS
AdoEnums.Schema.TABLEPRIVILEGES
AdoEnums.Schema.TABLES
AdoEnums.Schema.TRANSLATIONS
AdoEnums.Schema.TRUSTEES
AdoEnums.Schema.USAGEPRIVILEGES
AdoEnums.Schema.VIEWCOLUMNUSAGE
AdoEnums.Schema.VIEWS
AdoEnums.Schema.VIEWTABLEUSAGE

See Also

Applies To: [OpenSchema Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

SearchDirectionEnum

Specifies the direction of a record search within a [Recordset](#).

Constant	Value	Description
adSearchBackward	-1	Searches backward, stopping at the beginning of the Recordset . If a match is not found, the record pointer is positioned at BOF .
adSearchForward	1	Searches forward, stopping at the end of the Recordset . If a match is not found, the record pointer is positioned at EOF .

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.SearchDirection.BACKWARD

AdoEnums.SearchDirection.FORWARD

See Also

Applies To: [Find Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

SeekEnum

Specifies the type of [Seek](#) to execute.

Constant	Value	Description
adSeekFirstEQ	1	Seeks the first key equal to <i>KeyValues</i> .
adSeekLastEQ	2	Seeks the last key equal to <i>KeyValues</i> .
adSeekAfterEQ	4	Seeks either a key equal to <i>KeyValues</i> or just after where that match would have occurred.
adSeekAfter	8	Seeks a key just after where a match with <i>KeyValues</i> would have occurred.
adSeekBeforeEQ	16	Seeks either a key equal to <i>KeyValues</i> or just before where that match would have occurred.
adSeekBefore	32	Seeks a key just before where a match with <i>KeyValues</i> would have occurred.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.Seek.FIRSTEQ

AdoEnums.Seek.LASTEQ

AdoEnums.Seek.AFTEREQ

AdoEnums.Seek.AFTER

AdoEnums.Seek.BEFOREEQ

AdoEnums.Seek.BEFORE

See Also

Applies To: [Seek Method](#)

ADO 2.5 API Reference

StreamOpenOptionsEnum

Specifies options for opening a [Stream](#) object. The values can be combined with an OR operation.

Constant	Value	Description
adOpenStreamAsync	1	Opens the Stream object in asynchronous mode.
adOpenStreamFromRecord	4	Identifies the contents of the <i>Source</i> parameter to be an already open Record object. The default behavior is to treat <i>Source</i> as a URL that points directly to a node in a tree structure. The default stream associated with that node is opened.
adOpenStreamUnspecified	-1	Default. Specifies opening the Stream object with default options.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [Open Method \(ADO Stream\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

StreamReadEnum

Specifies whether the whole stream or the next line should be read from a [Stream](#) object.

Constant	Value	Description
adReadAll	-1	Default. Reads all bytes from the stream, from the current position onwards to the EOS marker. This is the only valid StreamReadEnum value with binary streams (Type is adTypeBinary).
adReadLine	-2	Reads the next line from the stream (designated by the LineSeparator property).

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [Read Method](#) | [ReadText Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

StreamTypeEnum

Specifies the type of data stored in a [Stream](#) object.

Constant	Value	Description
adTypeBinary	1	Indicates binary data.
adTypeText	2	Default. Indicates text data, which is in the character set specified by Charset .

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [Type Property \(ADO Stream\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

StreamWriteEnum

Specifies whether a line separator is appended to the string written to a [Stream](#) object.

Constant	Value	Description
adWriteChar	0	Default. Writes the specified text string (specified by the <i>Data</i> parameter) to the Stream object.
adWriteLine	1	Writes a text string and a line separator character to a Stream object. If the LineSeparator property is not defined, then this returns a run-time error.

ADO/WFC Equivalent

These constants do not have ADO/WFC equivalents.

See Also

Applies To: [WriteText Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

StringFormatEnum

Specifies the format when retrieving a [Recordset](#) as a string.

Constant	Value	Description
adClipString	2	Delimits rows by <i>RowDelimiter</i> , columns by <i>ColumnDelimiter</i> , and null values by <i>NullExpr</i> . These three parameters of the GetString method are valid only with a <i>StringFormat</i> of adClipString .

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.StringFormat.CLIPSTRING

See Also

Applies To: [GetString Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

XactAttributeEnum

Specifies the transaction attributes of a [Connection](#) object.

Constant	Value	Description
adXactAbortRetaining	262144	Performs retaining aborts—that is, calling RollbackTrans automatically starts a new transaction. Not all providers support this.
adXactCommitRetaining	131072	Performs retaining commits—that is, calling CommitTrans automatically starts a new transaction. Not all providers support this.

ADO/WFC Equivalent

Package: **com.ms.wfc.data**

Constant

AdoEnums.XactAttribute.ABORTRETAINING

AdoEnums.XactAttribute.COMMITRETAINING

See Also

Applies To: [Attributes Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO Syntax Indexes

The syntax for calling ADO methods and properties differs depending upon your development environment. The rest of the ADO Language Reference uses the Microsoft Visual Basic programming language to illustrate ADO method and property syntax. However, refer to the sections below for more specific syntax examples based on your programming language and methodology.

- The [ADO for Visual C++ Syntax Index for COM](#) covers ADO properties and methods without using the **#import** compiler directive with Microsoft Visual C++.
- The [ADO for Visual C++ Syntax Index with #import](#) covers ADO properties and methods when using the **#import** compiler directive with Microsoft Visual C++.
- The [ADO/WFC Syntax Index](#) covers using the ADO Windows Foundation Classes with Visual J++.

See Also

[Using ADO with Microsoft Visual Basic](#) | [Using ADO with Microsoft Visual C++](#) | [Using ADO with Microsoft Visual J++](#) | [Using ADO with Scripting Languages](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ADO for Visual C++ Syntax Index for COM

This index is a cross-reference to the ADO Language Reference based on Microsoft Visual C++.

If you use the **#import** directive in your application, a header file will be generated that will enable you to use syntax similar to Microsoft Visual Basic. Property names of the form **get_PropertyName** and **put_PropertyName** can be treated as if they were declared simply as *PropertyName*. A property can then be treated like a data member instead of a function.

All of the methods, properties, and events are functions that return an **HRESULT**, which you can test to determine if the function executed successfully.

Method and property syntax in Visual C++ is listed for the following elements:

- [Collections](#)
- [Command object](#)
- [Connection object](#)
- [Error object](#)
- [Field object](#)
- [Parameter object](#)
- [Record object](#)
- [Recordset object](#)
- [Stream object](#)

See Also

[ADO for Visual C++ Syntax Index with #import](#) | [ActiveX Data Objects Start Page](#)

ADO 2.5 API Reference

Collections (ADO for Visual C++ Syntax)

Parameters

Methods

[Append](#)(IDispatch *Object)

[Delete](#)(VARIANT Index)

[Refresh](#)(void)

Properties

[get_Count](#)(long *c)

[get_Item](#)(VARIANT Index, _ADOParameter **ppvObject)

Fields

Methods

[Append](#)(BSTR *bstrName*, DataTypeEnum *Type*, long *DefinedSize*, FieldAttr
[Delete](#)(VARIANT *Index*)
[Refresh](#)(void)

Properties

[get_Count](#)(long **c*)
[get_Item](#)(VARIANT *Index*, ADOField ***ppvObject*)

Errors

Methods

[Clear](#)(void)
[Refresh](#)(void)

Properties

[get_Count](#)(long *c)
[get_Item](#)(VARIANT *Index*, ADOError ***ppvObject*)

Properties

Methods

[Refresh](#)(void)

Properties

[get_Count](#)(long *c)

[get_Item](#)(VARIANT *Index*, ADOProperty ***ppvObject*)

See Also

[Errors Collection](#) | [Fields Collection](#) | [Parameters Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Command (ADO for Visual C++ Syntax)

Methods

[Cancel](#)(void)
[CreateParameter](#)(BSTR *Name*, DataTypeEnum *Type*, ParameterDirectionEnum
[Execute](#)(VARIANT **RecordsAffected*, VARIANT **Parameters*, long *Options*,

Properties

[get_ActiveConnection](#)(_ADOConnection ***ppvObject*)
[put_ActiveConnection](#)(VARIANT *vConn*)
[putref_ActiveConnection](#)(_ADOConnection **pCon*)
[get_CommandText](#)(BSTR **pbstr*)
[put_CommandText](#)(BSTR *bstr*)
[get_CommandTimeout](#)(LONG **pl*)
[put_CommandTimeout](#)(LONG *Timeout*)
[get_CommandType](#)(CommandTypeEnum **plCmdType*)
[put_CommandType](#)(CommandTypeEnum *lCmdType*)
[get_Name](#)(BSTR **pbstrName*)
[put_Name](#)(BSTR *bstrName*)
[get_Prepared](#)(VARIANT_BOOL **pfPrepared*)
[put_Prepared](#)(VARIANT_BOOL *fPrepared*)
[get_State](#)(LONG **pObjState*)
[get_Parameters](#)(ADOParameters ***ppvObject*)

See Also

[Command Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Connection (ADO for Visual C++ Syntax)

Methods

[BeginTrans](#)(long **TransactionLevel*)
[CommitTrans](#)(void)
[RollbackTrans](#)(void)
[Cancel](#)(void)
[Close](#)(void)
[Execute](#)(BSTR *CommandText*, VARIANT **RecordsAffected*, long *Options*, _AD
[Open](#)(BSTR *ConnectionString*, BSTR *UserID*, BSTR *Password*, long *Options*
[OpenSchema](#)(SchemaEnum *Schema*, VARIANT *Restrictions*, VARIANT *SchemaID*

Properties

[get_Attributes](#)(long **plAttr*)
[put_Attributes](#)(long *lAttr*)
[get_CommandTimeout](#)(LONG **plTimeout*)
[put_CommandTimeout](#)(LONG *lTimeout*)
[get_ConnectionString](#)(BSTR **pbstr*)
[put_ConnectionString](#)(BSTR *bstr*)
[get_ConnectionTimeout](#)(LONG **plTimeout*)
[put_ConnectionTimeout](#)(LONG *lTimeout*)
[get_CursorLocation](#)(CursorLocationEnum **plCursorLoc*)
[put_CursorLocation](#)(CursorLocationEnum *lCursorLoc*)
[get_DefaultDatabase](#)(BSTR **pbstr*)
[put_DefaultDatabase](#)(BSTR *bstr*)
[get_IsolationLevel](#)(IsolationLevelEnum **Level*)
[put_IsolationLevel](#)(IsolationLevelEnum *Level*)
[get_Mode](#)(ConnectModeEnum **plMode*)
[put_Mode](#)(ConnectModeEnum *lMode*)
[get_Provider](#)(BSTR **pbstr*)
[put_Provider](#)(BSTR *Provider*)
[get_State](#)(LONG **plobjState*)
[get_Version](#)(BSTR **pbstr*)
[get_Errors](#)(ADOErrors ***ppvObject*)

Events

[BeginTransComplete](#)(LONG *TransactionLevel*, ADOError **pError*,
EventStatusEnum **adStatus*, _ADODConnection **pConnec*
[CommitTransComplete](#)(ADOError **pError*, EventStatusEnum **adStatus*,

[_ADOConnection *pConnection\)](#)
[ConnectComplete](#)(ADOError *pError, EventStatusEnum *adStatus,
_ADOConnection *pConnection)
[Disconnect](#)(EventStatusEnum *adStatus, _ADOConnection *pConnection)
[ExecuteComplete](#)(LONG RecordsAffected, ADOError *pError,
EventStatusEnum *adStatus, _ADOCommand *pCommand,
_ADORecordset *pRecordset, _ADOConnection *pConnectio
[InfoMessage](#)(ADOError *pError, EventStatusEnum *adStatus,
_ADOConnection *pConnection)
[RollbackTransComplete](#)(ADOError *pError, EventStatusEnum *adStatus,
_ADOConnection *pConnection)
[WillConnect](#)(BSTR *ConnectionString, BSTR *UserID, BSTR *Password,
long *Options, EventStatusEnum *adStatus,
_ADOConnection *pConnection)
[WillExecute](#)(BSTR *Source, CursorTypeEnum *CursorType,
LockTypeEnum *LockType, long *Options,
EventStatusEnum *adStatus, _ADOCommand *pCommand,
_ADORecordset *pRecordset, _ADOConnection *pConnectio

See Also

[Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Error (ADO for Visual C++ Syntax)

Properties

[get_Description](#)(BSTR *pbstr)
[get_NativeError](#)(long *pl)
[get_Number](#)(long *pl)
[get_Source](#)(BSTR *pbstr)
[get_SQLState](#)(BSTR *pbstr)

See Also

[Error Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Field (ADO for Visual C++ Syntax)

Methods

[AppendChunk](#)(VARIANT *Data*)
[GetChunk](#)(long *Length*, VARIANT **pvar*)

Properties

[get_ActualSize](#)(long **pl*)
[get_Attributes](#)(long **pl*)
[put_Attributes](#)(long *lAttributes*)
[get_DataFormat](#)(IUnknown ***ppiDF*)
[put_DataFormat](#)(IUnknown **piDF*)
[get_DefinedSize](#)(long **pl*)
[put_DefinedSize](#)(long *lSize*)
[get_Name](#)(BSTR **pbstr*)
[get_NumericScale](#)(BYTE **pbNumericScale*)
[put_NumericScale](#)(BYTE *bScale*)
[get_OriginalValue](#)(VARIANT **pvar*)
[get_Precision](#)(BYTE **pbPrecision*)
[put_Precision](#)(BYTE *bPrecision*)
[get_Type](#)(DataTypeEnum **pDataType*)
[put_Type](#)(DataTypeEnum *DataType*)
[get_UnderlyingValue](#)(VARIANT **pvar*)
[get_Value](#)(VARIANT **pvar*)
[put_Value](#)(VARIANT *Val*)

See Also

[Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Parameter (ADO for Visual C++ Syntax)

Methods

[AppendChunk](#)(VARIANT *val*)

Properties

[get_Attributes](#)(LONG **plParmAttribs*)
[put_Attributes](#)(LONG *lParmAttribs*)
[get_Direction](#)(ParameterDirectionEnum **plParmDirection*)
[put_Direction](#)(ParameterDirectionEnum *lParmDirection*)
[get_Name](#)(BSTR **pbstr*)
[put_Name](#)(BSTR *bstr*)
[get_NumericScale](#)(BYTE **pbScale*)
[put_NumericScale](#)(BYTE *bScale*)
[get_Precision](#)(BYTE **pbPrecision*)
[put_Precision](#)(BYTE *bPrecision*)
[get_Size](#)(long **pl*)
[put_Size](#)(long *l*)
[get_Type](#)(DataTypeEnum **psDataType*)
[put_Type](#)(DataTypeEnum *sDataType*)
[get_Value](#)(VARIANT **pvar*)
[put_Value](#)(VARIANT *val*)

See Also

[Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Record (ADO for Visual C++ Syntax)

Methods

[Cancel](#)(void)

[Close](#)(void)

[CopyRecord](#)(BSTR *Source*, BSTR *Destination*, BSTR *UserName*, BSTR *Password*, CopyRecordOptionsEnum *Options*, VARIANT_BOOL *Async*, BSTR *pbstrNewURL*)

[DeleteRecord](#)(BSTR *Source*, VARIANT_BOOL *Async*)

[GetChildren](#)(_ADORecordset ***ppRSet*)

[MoveRecord](#)(BSTR *Source*, BSTR *Destination*, BSTR *UserName*, BSTR *Password*, MoveRecordOptionsEnum *Options*, VARIANT_BOOL *Async*, BSTR *pbstrNewURL*)

[Open](#)(VARIANT *Source*, VARIANT *ActiveConnection*, ConnectModeEnum *Mode*, RecordCreateOptionsEnum *CreateOptions*, RecordOpenOptionsEnum *Options*, BSTR *UserName*, BSTR *Password*)

Properties

[get_ActiveConnection](#)(VARIANT **pvar*)

[put_ActiveConnection](#)(BSTR *bstrConn*)

[putref_ActiveConnection](#)(_ADOConnection **Con*)

[get_Fields](#)(ADOFIELDS ***ppFlds*)

[get_Mode](#)(ConnectModeEnum **pMode*)

[put_Mode](#)(ConnectModeEnum *Mode*)

[get_ParentURL](#)(BSTR **pbstrParentURL*)

[get_RecordType](#)(RecordTypeEnum **pType*)

[get_Source](#)(VARIANT **pvar*)

[put_Source](#)(BSTR *Source*)

[putref_Source](#)(IDispatch **Source*)

[get_State](#)(ObjectStateEnum **pState*)

See Also

[Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Recordset (ADO for Visual C++ Syntax)

Methods

[AddNew](#)(VARIANT *FieldList*, VARIANT *Values*)
[Cancel](#)(void)
[CancelBatch](#)(AffectEnum *AffectRecords*)
[CancelUpdate](#)(void)
[Clone](#)(LockTypeEnum *LockType*, _ADORecordset ***ppvObject*)
[Close](#)(void)
[CompareBookmarks](#)(VARIANT *Bookmark1*, VARIANT *Bookmark2*, CompareEnum **pCompare*)
[Delete](#)(AffectEnum *AffectRecords*)
[Find](#)(BSTR *Criteria*, LONG *SkipRecords*, SearchDirectionEnum *SearchDire* VARIANT *Start*)
[GetRows](#)(long *Rows*, VARIANT *Start*, VARIANT *Fields*, VARIANT **pvar*)
[GetString](#)(StringFormatEnum *StringFormat*, long *NumRows*, BSTR *ColumnDe* BSTR *RowDelimiter*, BSTR *NullExpr*, BSTR **pRetString*)
[Move](#)(long *NumRecords*, VARIANT *Start*)
[MoveFirst](#)(void)
[MoveLast](#)(void)
[MoveNext](#)(void)
[MovePrevious](#)(void)
[NextRecordset](#)(VARIANT **RecordsAffected*, _ADORecordset ***ppiRs*)
[Open](#)(VARIANT *Source*, VARIANT *ActiveConnection*, CursorTypeEnum *Cursor* LockTypeEnum *LockType*, LONG *Options*)
[Requery](#)(LONG *Options*)
[Resync](#)(AffectEnum *AffectRecords*, ResyncEnum *ResyncValues*)
[Save](#)(BSTR *FileName*, PersistFormatEnum *PersistFormat*)
[Supports](#)(CursorOptionEnum *CursorOptions*, VARIANT_BOOL **pb*)
[Update](#)(VARIANT *Fields*, VARIANT *Values*)
[UpdateBatch](#)(AffectEnum *AffectRecords*)

Properties

[get_AbsolutePage](#)(PositionEnum **pI*)
[put_AbsolutePage](#)(PositionEnum *Page*)
[get_AbsolutePosition](#)(PositionEnum **pI*)
[put_AbsolutePosition](#)(PositionEnum *Position*)
[get_ActiveCommand](#)(IDispatch ***ppCmd*)
[get_ActiveConnection](#)(VARIANT **pvar*)
[put_ActiveConnection](#)(VARIANT *vConn*)

[putref_ActiveConnection](#)(IDispatch *pconn)
[get_BOF](#)(VARIANT_BOOL *pb)
[get_Bookmark](#)(VARIANT *pvBookmark)
[put_Bookmark](#)(VARIANT vBookmark)
[get_CacheSize](#)(long *pl)
[put_CacheSize](#)(long CacheSize)
[get_CursorLocation](#)(CursorLocationEnum *plCursorLoc)
[put_CursorLocation](#)(CursorLocationEnum lCursorLoc)
[get_CursorType](#)(CursorTypeEnum *plCursorType)
[put_CursorType](#)(CursorTypeEnum lCursorType)
[get_DataMember](#)(BSTR *pbstrDataMember)
[put_DataMember](#)(BSTR bstrDataMember)
[get_DataSource](#)(IUnknown **ppunkDataSource)
[putref_DataSource](#)(IUnknown *punkDataSource)
[get_EditMode](#)(EditModeEnum *pl)
[get_EOF](#)(VARIANT_BOOL *pb)
[get_Filter](#)(VARIANT *Criteria)
[put_Filter](#)(VARIANT Criteria)
[get_LockType](#)(LockTypeEnum *plLockType)
[put_LockType](#)(LockTypeEnum lLockType)
[get_MarshalOptions](#)(MarshalOptionsEnum *peMarshal)
[put_MarshalOptions](#)(MarshalOptionsEnum eMarshal)
[get_MaxRecords](#)(long *plMaxRecords)
[put_MaxRecords](#)(long lMaxRecords)
[get_PageCount](#)(long *pl)
[get_PageSize](#)(long *pl)
[put_PageSize](#)(long PageSize)
[get_RecordCount](#)(long *pl)
[get_Sort](#)(BSTR *Criteria)
[put_Sort](#)(BSTR Criteria)
[get_Source](#)(VARIANT *pvSource)
[put_Source](#)(BSTR bstrConn)
[putref_Source](#)(IDispatch *pcmd)
[get_State](#)(LONG *plObjState)
[get_Status](#)(long *pl)
[get_StayInSync](#)(VARIANT_BOOL *pbStayInSync)
[put_StayInSync](#)(VARIANT_BOOL bStayInSync)
[get_Fields](#)(ADOFIELDS **ppvObject)

Events

[EndOfRecordset](#)(VARIANT_BOOL *fMoreData, EventStatusEnum *adStatus, _ADORECORDSET *pRecordset)
[FetchComplete](#)(ADOError *pError, EventStatusEnum *adStatus, _ADORECORDSET *pRecordset)
[FetchProgress](#)(long Progress, long MaxProgress, EventStatusEnum *adStatus, _ADORECORDSET *pRecordset)
[FieldChangeComplete](#)(LONG cFields, VARIANT Fields, ADOError *pError, EventStatusEnum *adStatus, _ADORECORDSET *pRecords)

[MoveComplete](#)(EventReasonEnum *adReason*, ADOError **pError*,
EventStatusEnum **adStatus*, _ADORecordset **pRecords*)
[RecordChangeComplete](#)(EventReasonEnum *adReason*, LONG *cRecords*,
ADOError **pError*, EventStatusEnum **adStatus*,
_ADORecordset **pRecordset*)
[RecordsetChangeComplete](#)(EventReasonEnum *adReason*, ADOError **pError*,
EventStatusEnum **adStatus*, _ADORecordset **pRecords*)
[WillChangeField](#)(LONG *cFields*, VARIANT *Fields*, EventStatusEnum **adSta*
_ADORecordset **pRecordset*)
[WillChangeRecord](#)(EventReasonEnum *adReason*, LONG *cRecords*,
EventStatusEnum **adStatus*, _ADORecordset **pRecords*)
[WillChangeRecordset](#)(EventReasonEnum *adReason*, EventStatusEnum **adSta*
_ADORecordset **pRecordset*)
[WillMove](#)(EventReasonEnum *adReason*, EventStatusEnum **adStatus*,
_ADORecordset **pRecordset*)

See Also

[Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Stream (ADO for Visual C++ Syntax)

Methods

[Cancel](#)(void)
[Close](#)(void)
[CopyTo](#)(_ADOSTream *DestStream, LONG CharNumber = -1)
[Flush](#)(void)
[LoadFromFile](#)(BSTR FileName)
[Open](#)(VARIANT Source, ConnectModeEnum Mode, StreamOpenOptionsEnum Opt
BSTR UserName, BSTR Password)
[Read](#)(long NumBytes, VARIANT *pVal)
[ReadText](#)(long NumChars, BSTR *pbstr)
[SaveToFile](#)(BSTR FileName, SaveOptionsEnum Options = adSaveCreateNotE
[SetEOS](#)(void)
[SkipLine](#)(void)
[Write](#)(VARIANT Buffer)
[WriteText](#)(BSTR Data, StreamWriteEnum Options = adWriteChar)

Properties

[get_Charset](#)(BSTR *pbstrCharset)
[put_Charset](#)(BSTR Charset)
[get_EOS](#)(VARIANT_BOOL *pEOS)
[get_LineSeparator](#)(LineSeparatorEnum *pLS)
[put_LineSeparator](#)(LineSeparatorEnum LineSeparator)
[get_Mode](#)(ConnectModeEnum *pMode)
[put_Mode](#)(ConnectModeEnum Mode)
[get_Position](#)(LONG *pPos)
[put_Position](#)(LONG Position)
[get_Size](#)(LONG *pSize)
[get_State](#)(ObjectStateEnum *pState)
[get_Type](#)(StreamTypeEnum *pType)
[put_Type](#)(StreamTypeEnum Type)

See Also

[Stream Object](#)

ADO 2.5 API Reference

ADO for Visual C++ Syntax Index with #import

This index is a cross-reference to the ADO Language Reference based on Microsoft Visual C++ and the **#import** directive.

This particular index was derived by compiling a program with the **#import** directive against the ADO .dll, then reformatting the *.tlh file that was generated. Only information about methods, properties, and events was preserved. The alternative syntax declared for each property is listed by the corresponding "__declspec(property...)" directive.

You are strongly encouraged to read [Visual C++ ADO Programming](#) for more information.

Method and property syntax in Visual C++ with the **#import** directive is listed for the following elements:

- [ADO Collections](#)
- [Command Object](#)
- [Connection Object](#)
- [Error Object](#)
- [Field Object](#)
- [Parameter Object](#)
- [Property object](#)
- [Record Object](#)
- [Recordset Object](#)
- [Stream Object](#)
- [Connection Events](#)
- [Recordset Events](#)

See Also

[ADO for Visual C++ Syntax Index for COM](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Collections (Visual C++ Syntax Index with #import)

It is useful to know that collections inherit certain common methods and properties.

All collections inherit the **Count** property and **Refresh** method, and all collections add the **Item** property. The **Errors** collection adds the **Clear** method. The **Parameters** collection inherits the **Append** and **Delete** methods, while the **Fields** collection adds the **Append**, **Delete**, and **Update** methods.

Properties Collection

Methods

```
HRESULT Refresh( );
```

Properties

```
long GetCount( );  
__declspec(property(get=GetCount)) long Count;
```

```
PropertyPtr GetItem( const _variant_t & Index );  
__declspec(property(get=GetItem)) PropertyPtr Item[];
```

Errors Collection

Methods

HRESULT [Clear](#)();

HRESULT [Refresh](#)();

Properties

long **GetCount**();
__declspec(property(get=GetCount)) long [Count](#);

PropertyPtr **GetItem**(const _variant_t & *Index*);
__declspec(property(get=GetItem)) PropertyPtr [Item](#)[];

Parameters Collection

Methods

HRESULT [Append](#)(IDispatch * *Object*);

HRESULT [Delete](#)(const _variant_t & *Index*);

HRESULT [Refresh](#)();

Properties

long **GetCount**();
__declspec(property(get=GetCount)) long [Count](#);

PropertyPtr **GetItem**(const _variant_t & *Index*);
__declspec(property(get=GetItem)) PropertyPtr [Item](#)[];

Fields Collection

Methods

```
HRESULT Append( _bstr_t Name, enum DataTypeEnum Type, long DefinedSi  
    enum FieldAttributeEnum Attrib, const _variant_t & FieldValue =  
    vtMissing );
```

```
HRESULT Delete( const _variant_t & Index );
```

```
HRESULT Refresh( );
```

```
HRESULT Update( );
```

Properties

```
long GetCount( );  
__declspec(property(get=GetCount)) long Count;
```

```
PropertyPtr GetItem( const _variant_t & Index );  
__declspec(property(get=GetItem)) PropertyPtr Item[];
```

See Also

[Errors Collection](#) | [Fields Collection](#) | [Parameters Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Command (Visual C++ Syntax Index with #import)

Methods

```
HRESULT Cancel( );
```

```
_RecordsetPtr Execute( VARIANT * RecordsAffected, VARIANT  
* Parameters, long Options );
```

```
_ParameterPtr CreateParameter( _bstr_t Name, enum  
DataTypeEnum Type, enum ParameterDirectionEnum Direction, long S  
const _variant_t & Value = vtMissing );
```

Properties

```
_ConnectionPtr GetActiveConnection( );  
void PutRefActiveConnection( struct _Connection * ppvObject );  
void PutActiveConnection( const _variant_t & ppvObject );  
__declspec(property(get=GetActiveConnection,put=PutRefActiveConnecti  
_ConnectionPtr ActiveConnection;
```

```
_bstr_t GetCommandText( );  
void PutCommandText( _bstr_t pbstr );  
__declspec(property(get=GetCommandText,put=PutCommandText)) _bstr_t  
CommandText;
```

```
long GetCommandTimeout( );  
void PutCommandTimeout( long pl );  
__declspec(property(get=GetCommandTimeout,put=PutCommandTimeout)) lo  
CommandTimeout;
```

```
void PutCommandType( enum CommandTypeEnum plCmdType );  
enum CommandTypeEnum GetCommandType( );  
__declspec(property(get=GetCommandType,put=PutCommandType)) enum  
CommandTypeEnum CommandType;
```

```
VARIANT_BOOL GetPrepared( );  
void PutPrepared( VARIANT_BOOL pfPrepared );  
__declspec(property(get=GetPrepared,put=PutPrepared)) VARIANT_BOOL  
Prepared;
```

```
ParametersPtr GetParameters( );
```

```
__declspec(property(get=GetParameters)) ParametersPtr  
    Parameters;  
  
_bstr_t GetName( );  
void PutName( _bstr_t pbstrName );  
__declspec(property(get=GetName,put=PutName)) _bstr_t Name;  
  
long GetState( );  
__declspec(property(get=GetState)) long State;
```

See Also

[Command Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Connection (Visual C++ Syntax Index with #import)

Methods

```
HRESULT Cancel( );
```

```
HRESULT Close( );
```

```
_RecordsetPtr Execute( _bstr_t CommandText, VARIANT *  
    RecordsAffected, long Options );
```

```
long BeginTrans( );
```

```
HRESULT CommitTrans( );
```

```
HRESULT RollbackTrans( );
```

```
HRESULT Open( _bstr_t ConnectionString, _bstr_t UserID,  
    _bstr_t Password, long Options );
```

```
_RecordsetPtr OpenSchema( enum SchemaEnum Schema, const  
    _variant_t & Restrictions = vtMissing, const _variant_t &  
    SchemaID = vtMissing );
```

Properties

```
_bstr_t GetConnectionString( );
```

```
void PutConnectionString( _bstr_t pbstr );
```

```
__declspec(property(get=GetConnectionString,put=PutConnectionString))  
    _bstr_t ConnectionString;
```

```
long GetCommandTimeout( );
```

```
void PutCommandTimeout( long pTimeout );
```

```
__declspec(property(get=GetCommandTimeout,put=PutCommandTimeout)) long  
    CommandTimeout;
```

```
long GetConnectionTimeout( );
```

```
void PutConnectionTimeout( long pTimeout );
```

```
__declspec(property(get=GetConnectionTimeout,put=PutConnectionTimeou  
    long ConnectionTimeout;
```

```
_bstr_t GetVersion( );
```

```
__declspec(property(get=GetVersion)) _bstr_t Version;
```

```

ErrorsPtr GetErrors( );
__declspec(property(get=GetErrors)) ErrorsPtr Errors;

_bstr_t GetDefaultDatabase( );
void PutDefaultDatabase( _bstr_t pbstr );
__declspec(property(get=GetDefaultDatabase,put=PutDefaultDatabase))
    _bstr_t DefaultDatabase;

enum IsolationLevelEnum GetIsolationLevel( );
void PutIsolationLevel( enum IsolationLevelEnum Level );
__declspec(property(get=GetIsolationLevel,put=PutIsolationLevel)) en
    IsolationLevelEnum IsolationLevel;

long GetAttributes( );
void PutAttributes( long pIAttr );
__declspec(property(get=GetAttributes,put=PutAttributes)) long
    Attributes;

enum CursorLocationEnum GetCursorLocation( );
void PutCursorLocation( enum CursorLocationEnum pICursorLoc );
__declspec(property(get=GetCursorLocation,put=PutCursorLocation)) en
    CursorLocationEnum CursorLocation;

enum ConnectModeEnum GetMode( );
void PutMode( enum ConnectModeEnum pIMode );
__declspec(property(get=GetMode,put=PutMode)) enum ConnectModeEnum
    Mode;

_bstr_t GetProvider( );
void PutProvider( _bstr_t pbstr );
__declspec(property(get=GetProvider,put=PutProvider)) _bstr_t
    Provider;

long GetState( );
__declspec(property(get=GetState)) long State;

```

See Also

[Connection Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Error (Visual C++ Syntax Index with #import)

Properties

```
_bstr_t GetDescription( );  
__declspec(property(get=GetDescription)) _bstr_t Description;  
  
long GetHelpContext( );  
__declspec(property(get=GetHelpContext)) long HelpContext;  
  
_bstr_t GetHelpFile( );  
__declspec(property(get=GetHelpFile)) _bstr_t HelpFile;  
  
long GetNativeError( );  
__declspec(property(get=GetNativeError)) long NativeError;  
  
long GetNumber( );  
__declspec(property(get=GetNumber)) long Number;  
  
_bstr_t GetSource( );  
__declspec(property(get=GetSource)) _bstr_t Source;  
  
_bstr_t GetSQLState( );  
__declspec(property(get=GetSQLState)) _bstr_t SQLState;
```

See Also

[Error Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Field (Visual C++ Syntax Index with #import)

Methods

```
HRESULT AppendChunk( const _variant_t & Data );
```

```
_variant_t GetChunk( long Length );
```

Properties

```
long GetActualSize( );  
__declspec(property(get=GetActualSize)) long ActualSize;
```

```
long GetAttributes( );  
void PutAttributes( long p1 );  
__declspec(property(get=GetAttributes,put=PutAttributes)) long A
```

```
IUnknownPtr GetDataFormat( );  
void PutRefDataFormat( IUnknown * ppiDF );  
__declspec(property(get=GetDataFormat,put=PutRefDataFormat)) IUnknown  
DataFormat;
```

```
long GetDefinedSize( );  
void PutDefinedSize( long p1 );  
__declspec(property(get=GetDefinedSize,put=PutDefinedSize)) long  
DefinedSize;
```

```
_bstr_t GetName( );  
__declspec(property(get=GetName)) _bstr_t Name;
```

```
unsigned char GetNumericScale( );  
void PutNumericScale( unsigned char pbNumericScale );  
__declspec(property(get=GetNumericScale,put=PutNumericScale)) unsigned  
char NumericScale;
```

```
_variant_t GetOriginalValue( );  
__declspec(property(get=GetOriginalValue)) _variant_t OriginalValue;
```

```
unsigned char GetPrecision( );  
void PutPrecision( unsigned char pbPrecision );  
__declspec(property(get=GetPrecision,put=PutPrecision)) unsigned cha  
Precision;
```

```
enum DataTypeEnum GetType( );  
void PutType( enum DataTypeEnum pDataType );  
__declspec(property(get=GetType,put=PutType)) enum DataTypeEnum Type  
  
_variant_t GetUnderlyingValue( );  
__declspec(property(get=GetUnderlyingValue)) _variant_t UnderlyingVa  
  
_variant_t GetValue( );  
void PutValue( const _variant_t & pvar );  
__declspec(property(get=GetValue,put=PutValue)) _variant_t Value;
```

See Also

[Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Parameter (Visual C++ Syntax Index with #import)

Methods

```
HRESULT AppendChunk( const _variant_t & Val );
```

Properties

```
long GetAttributes( );  
void PutAttributes( long pIParamAttribs );  
__declspec(property(get=GetAttributes,put=PutAttributes)) long  
Attributes;
```

```
enum ParameterDirectionEnum GetDirection( );  
void PutDirection( enum ParameterDirectionEnum pIParamDirection );  
__declspec(property(get=GetDirection,put=PutDirection)) enum  
ParameterDirectionEnum Direction;
```

```
_bstr_t GetName( );  
void PutName( _bstr_t pbstr );  
__declspec(property(get=GetName,put=PutName)) _bstr_t Name;
```

```
unsigned char GetNumericScale( );  
void PutNumericScale( unsigned char pbScale );  
__declspec(property(get=GetNumericScale,put=PutNumericScale)) unsign  
char NumericScale;
```

```
unsigned char GetPrecision( );  
void PutPrecision( unsigned char pbPrecision );  
__declspec(property(get=GetPrecision,put=PutPrecision)) unsigned cha  
Precision;
```

```
long GetSize( );  
void PutSize( long pI );  
__declspec(property(get=GetSize,put=PutSize)) long Size;
```

```
enum DataTypeEnum GetType( );  
void PutType( enum DataTypeEnum psDataType );  
__declspec(property(get=GetType,put=PutType)) enum DataTypeEnum Type
```

```
_variant_t GetValue( );  
void PutValue( const _variant_t & pvar );
```

`__declspec(property(get=GetValue,put=PutValue)) _variant_t Value;`

See Also

[Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Property (Visual C++ Syntax Index with #import)

Properties

```
long GetAttributes( );  
void PutAttributes( long pAttributes );  
__declspec(property(get=GetAttributes,put=PutAttributes)) long  
Attributes;  
  
_bstr_t GetName( );  
__declspec(property(get=GetName)) _bstr_t Name;  
  
enum DataTypeEnum GetType( );  
__declspec(property(get=GetType)) enum DataTypeEnum Type;  
  
_variant_t GetValue( );  
void PutValue( const _variant_t & pval );  
__declspec(property(get=GetValue,put=PutValue)) _variant_t Value;
```

See Also

[Property Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Record (Visual C++ Syntax Index with #import)

Methods

HRESULT [Cancel](#)();

HRESULT [Close](#)();

_bstr_t [CopyRecord](#)(*_bstr_t Source*, *_bstr_t Destination*,
_bstr_t UserName, *_bstr_t Password*, enum CopyRecordOptionsEn
Options, VARIANT_BOOL *Async*);

HRESULT [DeleteRecord](#)(*_bstr_t Source*, VARIANT_BOOL *Async*);

_RecordsetPtr [GetChildren](#)();

_bstr_t [MoveRecord](#)(*_bstr_t Source*, *_bstr_t Destination*,
_bstr_t UserName, *_bstr_t Password*, enum MoveRecordOptionsEn
Options, VARIANT_BOOL *Async*);

HRESULT [Open](#)(const *_variant_t & Source*, const *_variant_t*
& *ActiveConnection*, enum ConnectModeEnum *Mode*, enum
RecordCreateOptionsEnum *CreateOptions*, enum RecordOpenOptionsEnu
Options, *_bstr_t UserName*, *_bstr_t Password*);

Properties

_variant_t [GetActiveConnection](#)();
void **PutActiveConnection**(*_bstr_t pvar*);
void **PutRefActiveConnection**(struct *_Connection * pvar*);

FieldsPtr **GetFields**();
__declspec(property(get=GetFields)) *FieldsPtr* [Fields](#);

enum ConnectModeEnum **GetMode**();
void **PutMode**(enum ConnectModeEnum *pMode*);
__declspec(property(get=GetMode,put=PutMode)) enum ConnectModeEnum [M](#)

_bstr_t **GetParentURL**();
__declspec(property(get=GetParentURL)) *_bstr_t* [ParentURL](#);

enum RecordTypeEnum **GetRecordType**();

```
__declspec(property(get=GetRecordType)) enum RecordTypeEnum  
RecordType;  
  
_variant_t GetSource( );  
void PutSource( _bstr_t pvar );  
void PutRefSource( IDispatch * pvar );  
  
enum ObjectStateEnum GetState( );  
__declspec(property(get=GetState)) enum ObjectStateEnum State;
```

See Also

[Record Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Recordset (Visual C++ Syntax Index with #import)

Methods

```
HRESULT AddNew( const _variant_t & FieldList = vtMissing,
    const _variant_t & Values = vtMissing );

HRESULT Cancel( );

HRESULT CancelBatch( enum AffectEnum AffectRecords );

HRESULT CancelUpdate( );

_RecordsetPtr Clone( enum LockTypeEnum LockType );

HRESULT Close( );

enum CompareEnum CompareBookmarks( const _variant_t
    & Bookmark1, const _variant_t & Bookmark2 );

HRESULT Delete( enum AffectEnum AffectRecords );

HRESULT Find( _bstr_t Criteria, long SkipRecords, enum
    SearchDirectionEnum SearchDirection, const _variant_t & Start =
    vtMissing );

_variant_t GetRows( long Rows, const _variant_t & Start =
    vtMissing, const _variant_t & Fields = vtMissing );

_bstr_t GetString( enum
    StringFormatEnum StringFormat, long NumRows, _bstr_t
    ColumnDelimiter, _bstr_t RowDelimiter, _bstr_t NullExpr );

HRESULT Move( long NumRecords, const _variant_t & Start =
    vtMissing );

HRESULT MoveFirst( );
HRESULT MoveLast( );
HRESULT MoveNext( );
HRESULT MovePrevious( );

_RecordsetPtr NextRecordset( VARIANT * RecordsAffected );
```

```

HRESULT Open( const _variant_t & Source, const _variant_t &
    ActiveConnection, enum CursorTypeEnum CursorType, enum LockTypeE
    LockType, long Options );

HRESULT Requery( long Options );

HRESULT Update( const _variant_t & Fields = vtMissing, const
    _variant_t & Values = vtMissing );

HRESULT UpdateBatch( enum AffectEnum AffectRecords );

HRESULT Resync( enum AffectEnum AffectRecords, enum
    ResyncEnum ResyncValues );

HRESULT Save( const _variant_t & Destination, enum
    PersistFormatEnum PersistFormat );

HRESULT Seek( const _variant_t & KeyValues, enum SeekEnum
    SeekOption );

VARIANT_BOOL Supports( enum CursorOptionEnum CursorOptions );

```

Properties

```

enum PositionEnum GetAbsolutePage( );
void PutAbsolutePage( enum PositionEnum p1 );
__declspec(property(get=GetAbsolutePage,put=PutAbsolutePage)) enum
    PositionEnum AbsolutePage;

enum PositionEnum GetAbsolutePosition( );
void PutAbsolutePosition( enum PositionEnum p1 );
__declspec(property(get=GetAbsolutePosition,put=PutAbsolutePosition))
    enum PositionEnum AbsolutePosition;

IDispatchPtr GetActiveCommand( );
__declspec(property(get=GetActiveCommand)) IDispatchPtr ActiveComman

void PutRefActiveConnection( IDispatch * pvar );
void PutActiveConnection( const _variant_t & pvar );
_variant_t GetActiveConnection( );

enum CursorLocationEnum GetCursorLocation( );
void PutCursorLocation( enum CursorLocationEnum p1CursorLoc );
__declspec(property(get=GetCursorLocation,put=PutCursorLocation)) en
    CursorLocationEnum CursorLocation;

VARIANT_BOOL GetBOF( );
__declspec(property(get=GetBOF)) VARIANT_BOOL BOF;

```

```

VARIANT_BOOL GetEndOfFile( ); // Actually, GetEOF. Renamed in #import
__declspec(property(get=GetEndOfFile)) VARIANT_BOOL EndOfFile;

_variant_t GetBookmark( );
void PutBookmark( const _variant_t & pvBookmark );
__declspec(property(get=GetBookmark,put=PutBookmark)) _variant_t
Bookmark;

long GetCacheSize( );
void PutCacheSize( long pI );
__declspec(property(get=GetCacheSize,put=PutCacheSize)) long
CacheSize;

enum CursorTypeEnum GetCursorType( );
void PutCursorType( enum CursorTypeEnum pICursorType );
__declspec(property(get=GetCursorType,put=PutCursorType)) enum
CursorTypeEnum CursorType;

_bstr_t GetDataMember( );
void PutDataMember( _bstr_t pbstrDataMember );
__declspec(property(get=GetDataMember,put=PutDataMember)) _bstr_t
DataMember;

IUnknownPtr GetDataSource( );
void PutRefDataSource( IUnknown * ppunkDataSource );
__declspec(property(get=GetDataSource,put=PutRefDataSource)) IUnknow
DataSource;

enum EditModeEnum GetEditMode( );
__declspec(property(get=GetEditMode)) enum EditModeEnum EditMode;

FieldsPtr GetFields( );
__declspec(property(get=GetFields)) FieldsPtr Fields;

_variant_t GetFilter( );
void PutFilter( const _variant_t & Criteria );
__declspec(property(get=GetFilter,put=PutFilter)) _variant_t Filter;

_bstr_t GetIndex( );
void PutIndex( _bstr_t pbstrIndex );
__declspec(property(get=GetIndex,put=PutIndex)) _bstr_t Index;

enum LockTypeEnum GetLockType( );
void PutLockType( enum LockTypeEnum pILockType );
__declspec(property(get=GetLockType,put=PutLockType)) enum LockTypeE
LockType;

enum MarshalOptionsEnum GetMarshalOptions( );
void PutMarshalOptions( enum MarshalOptionsEnum peMarshal );
__declspec(property(get=GetMarshalOptions,put=PutMarshalOptions)) en

```

```

    MarshalOptionsEnum MarshalOptions;

long GetMaxRecords( );
void PutMaxRecords( long p1MaxRecords );
__declspec(property(get=GetMaxRecords,put=PutMaxRecords)) long
    MaxRecords;

long GetPageCount( );
__declspec(property(get=GetPageCount)) long PageCount;

long GetPageSize( );
void PutPageSize( long p1 );
__declspec(property(get=GetPageSize,put=PutPageSize)) long PageSize;

long GetRecordCount( );
__declspec(property(get=GetRecordCount)) long RecordCount;

_bstr_t GetSort( );
void PutSort( _bstr_t Criteria );
__declspec(property(get=GetSort,put=PutSort)) _bstr_t Sort;

void PutRefSource( IDispatch * pvSource );
void PutSource( _bstr_t pvSource );
_variant_t GetSource( );

long GetState( );
__declspec(property(get=GetState)) long State;

long GetStatus( );
__declspec(property(get=GetStatus)) long Status;

VARIANT_BOOL GetStayInSync( );
void PutStayInSync( VARIANT_BOOL pbStayInSync );
__declspec(property(get=GetStayInSync,put=PutStayInSync)) VARIANT_BOOL
    StayInSync;

```

See Also

[Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Stream (Visual C++ Syntax Index with #import)

Methods

```
HRESULT Cancel( );  
  
HRESULT Close( );  
  
HRESULT CopyTo( struct _Stream * DestStream, int CharNumber );  
  
HRESULT Flush( );  
  
HRESULT LoadFromFile( _bstr_t FileName );  
  
HRESULT Open( const _variant_t & Source, enum  
    ConnectModeEnum Mode, enum    StreamOpenOptionsEnum Options, _b  
    UserName, _bstr_t Password );  
  
_variant_t Read( long NumBytes );  
  
_bstr_t ReadText( long NumChars );  
  
HRESULT SaveToFile( _bstr_t FileName, enum SaveOptionsEnum  
    Options );  
  
HRESULT SetEOS( );  
  
HRESULT SkipLine( );  
  
HRESULT Write( const _variant_t & Buffer );  
  
HRESULT WriteText( _bstr_t Data, enum StreamWriteEnum  
    Options );
```

Properties

```
_bstr_t GetCharset( );  
void PutCharset( _bstr_t pbstrCharset );  
__declspec(property(get=GetCharset,put=PutCharset)) _bstr_t Charset;  
  
VARIANT_BOOL GetEOS( );  
__declspec(property(get=GetEOS)) VARIANT_BOOL EOS;
```

```
enum LineSeparatorEnum GetLineSeparator( );  
void PutLineSeparator( enum LineSeparatorEnum pLS );  
__declspec(property(get=GetLineSeparator,put=PutLineSeparator)) enum  
LineSeparatorEnum LineSeparator;
```

```
enum ConnectModeEnum GetMode( );  
void PutMode( enum ConnectModeEnum pMode );  
__declspec(property(get=GetMode,put=PutMode)) enum ConnectModeEnum M
```

```
long GetPosition( );  
void PutPosition( long pPos );  
__declspec(property(get=GetPosition,put=PutPosition)) long Position;
```

```
long GetSize( );  
__declspec(property(get=GetSize)) long Size;
```

```
enum ObjectStateEnum GetState( );  
__declspec(property(get=GetState)) enum ObjectStateEnum State;
```

```
enum StreamTypeEnum GetType( );  
void PutType( enum StreamTypeEnum pType );  
__declspec(property(get=GetType,put=PutType)) enum StreamTypeEnum Ty
```

See Also

[Stream Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

ConnectionEvents (Visual C++ Syntax Index with #import)

Events

```
HRESULT InfoMessage( struct Error * pError, enum  
    EventStatusEnum * adStatus, struct _Connection * pConnection
```

```
HRESULT BeginTransComplete( long TransactionLevel,  
    struct Error * pError, enum EventStatusEnum * adStatus, struct  
    _Connection * pConnection );
```

```
HRESULT CommitTransComplete( struct Error *  
    pError, enum EventStatusEnum * adStatus, struct _Connection  
    pConnection );
```

```
HRESULT RollbackTransComplete( struct Error *  
    pError, enum EventStatusEnum * adStatus, struct _Connection  
    pConnection );
```

```
HRESULT WillExecute( BSTR * Source, enum  
    CursorTypeEnum * CursorType,  
    enum LockTypeEnum * LockType, long * Options, enum EventStatusEn  
    adStatus, struct _Command * pCommand, struct _Recordset * pRecor  
    struct _Connection * pConnection );
```

```
HRESULT ExecuteComplete( long RecordsAffected, struct  
    Error * pError, enum EventStatusEnum * adStatus, struct _Com  
    * pCommand, struct _Recordset * pRecordset, struct _Connecti  
    pConnection );
```

```
HRESULT WillConnect( BSTR * ConnectionString, BSTR *  
    UserID, BSTR * Password, long * Options, enum EventStatusEnu  
    adStatus, struct _Connection * pConnection );
```

```
HRESULT ConnectComplete( struct Error *  
    pError, enum EventStatusEnum * adStatus, struct _Connection  
    pConnection );
```

```
HRESULT Disconnect( enum EventStatusEnum *  
    adStatus, struct _Connection * pConnection );
```

ADO 2.5 API Reference

RecordsetEvents (Visual C++ Syntax Index with #import)

Events

```
HRESULT WillChangeField( long cFields, const  
    _variant_t & Fields, enum      EventStatusEnum * adStatus, struct  
    _Recordset * pRecordset );
```

```
HRESULT FieldChangeComplete( long cFields, const  
    _variant_t & Fields,      struct Error * pError, enum EventStatus  
    * adStatus, struct      _Recordset * pRecordset );
```

```
HRESULT WillChangeRecord( enum EventReasonEnum  
    adReason, long cRecords,      enum EventStatusEnum * adStatus, st  
    _Recordset * pRecordset );
```

```
HRESULT RecordChangeComplete( enum EventReasonEnum  
    adReason, long      cRecords, struct Error * pError, enum  
    EventStatusEnum * adStatus,      struct _Recordset * pRecordset )
```

```
HRESULT WillChangeRecordset( enum EventReasonEnum  
    adReason, enum      EventStatusEnum * adStatus, struct _Recordset  
    pRecordset );
```

```
HRESULT RecordsetChangeComplete( enum  
    EventReasonEnum adReason, struct      Error * pError, enum  
    EventStatusEnum * adStatus, struct _Recordset *      pRecordset )
```

```
HRESULT WillMove( enum EventReasonEnum adReason, enum  
    EventStatusEnum *      adStatus, struct _Recordset * pRecordset )
```

```
HRESULT MoveComplete( enum EventReasonEnum adReason, struct  
    Error *      pError, enum EventStatusEnum * adStatus, struct  
    _Recordset * pRecordset );
```

```
HRESULT EndOfRecordset( VARIANT_BOOL * fMoreData, enum  
    EventStatusEnum *      adStatus, struct _Recordset * pRecordset )
```

```
HRESULT FetchProgress( long Progress, long MaxProgress,  
    enum      EventStatusEnum * adStatus, struct _Recordset * pRecordset)
```

```
HRESULT FetchComplete( struct Error * pError, enum  
    EventStatusEnum *      adStatus, struct _Recordset * pRecordset )
```

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

ADO/WFC Syntax Index

The ADO Language Reference uses the Microsoft Visual Basic programming language to illustrate ADO method and property syntax. This index is a cross-reference to the ADO Language Reference topics, based on Microsoft Visual J++ and ADO for Windows Foundation Classes (ADO/WFC). When differences in syntax arise, use the function signatures in this index, as opposed to the syntax listings in the language reference topic.

Method and property syntax are listed for the following elements:

ActiveX Data Objects

- [ADO Collections](#)
- [Command object](#)
- [Connection object](#)
- [Error object](#)
- [Field object](#)
- [Parameter object](#)
- [Recordset object](#)

Remote Data Service

- [DataSpace](#)
- [ObjectProxy](#)

See Also

[ADO Event Model, Synchronous and Asynchronous Operations](#) | [ADO for Windows Foundation Classes](#) | [ActiveX Data Objects Start Page](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Collections (ADO/WFC Syntax)

package com.ms.wfc.data

Parameters

Methods

```
public void append(com.ms.wfc.data.Parameter param)  
public void delete(int n)  
public void delete(String s)  
public void refresh()  
public Parameter getItem(int n)  
public Parameter getItem(String s)
```

Properties

```
public int getCount()
```

Fields

Methods

```
public void append(String name, int type)
public void append(String name, int type, int definedSize)
public void append(String name, int type, int definedSize, int attri
public void delete(int n)
public void delete(String s)
public void refresh()
public com.ms.wfc.data.Field getItem(int n)
public com.ms.wfc.data.Field getItem(String s)
```

Properties

```
public int getCount()
```

Errors

Methods

```
public void clear()  
public void refresh()  
public com.ms.wfc.data.Error getItem(int n)  
public com.ms.wfc.data.Error getItem(String s)
```

Properties

```
public int getCount()
```

See Also

[Errors Collection](#) | [Fields Collection](#) | [Parameters Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Command (ADO/WFC Syntax)

```
package com.ms.wfc.data
```

Constructor

```
public Command()  
public Command(String commandtext)
```

Methods

```
public void cancel()  
public com.ms.wfc.data.Parameter createParameter(String  
    Name, int Type, int Direction, int Size, Object Value)  
public Recordset execute()  
public Recordset execute(Object[] parameters)  
public Recordset execute(Object[] parameters, int options)  
public int executeUpdate(Object[] parameters)  
public int executeUpdate(Object[] parameters, int options)  
public int executeUpdate()
```

The **executeUpdate** method is a special case method that calls the underlying ADO **execute** method with certain parameters. The **executeUpdate** method does not support the return of a **Recordset** object, so the **execute** method's *options* parameter is modified with **AdoEnums.ExecuteOptions.NORECORDS**. After the **execute** method completes, its updated *RecordsAffected* parameter is passed back to the **executeUpdate** method, which is finally returned as an **int**.

Properties

```
public com.ms.wfc.data.Connection getActiveConnection()  
public void setActiveConnection(com.ms.wfc.data.Connection con)  
public void setActiveConnection(String conString)  
public String getCommandText()  
public void setCommandText(String command)  
public int getCommandTimeout()  
public void setCommandTimeout(int timeout)  
public int getCommandType()  
public void setCommandType(int type)  
public String getName()  
public void setName(String name)  
public boolean getPrepared()
```

```
public void setPrepared(boolean prepared)  
public int getState()  
public com.ms.wfc.data.Parameter getParameter(int n)  
public com.ms.wfc.data.Parameter getParameter(String n)  
public com.ms.wfc.data.Parameters getParameters()  
public AdoProperties getProperties()
```

See Also

[Command Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Connection (ADO/WFC Syntax)

```
package com.ms.wfc.data
```

Constructor

```
public Connection()  
public Connection(String connectionstring)
```

Methods

```
public int beginTrans()  
public void commitTrans()  
public void rollbackTrans()  
public void cancel()  
public void close()  
public com.ms.wfc.data.Recordset execute(String commandText)  
public com.ms.wfc.data.Recordset execute(String commandText, int opt)  
public int executeUpdate(String commandText)  
public int executeUpdate(String commandText, int options)
```

The **executeUpdate** method is a special case method that calls the underlying ADO **execute** method with certain parameters. The **executeUpdate** method does not support the return of a **Recordset** object, so the **execute** method's *options* parameter is modified with **AdoEnums.ExecuteOptions.NO RECORDS**. After the **execute** method completes, its updated *RecordsAffected* parameter is passed back to the **executeUpdate** method, which is finally returned as an **int**.

```
public void open()  
public void open(String connectionString)  
public void open(String connectionString, String userID)  
public void open(String connectionString, String userID, String pass)  
public void open(String connectionString, String userID, String pass)  
public Recordset openSchema(int schema, Object[]  
    restrictions, String schemaID)  
public Recordset openSchema(int schema)  
public Recordset openSchema(int schema, Object[] restrictions)
```

Properties

```
public int getAttributes()  
public void setAttributes(int attr)
```

```

public int getCommandTimeout()
public void setCommandTimeout(int timeout)
public String getConnectionString()
public void setConnectionString(String con)
public int getConnectionTimeout()
public void setConnectionTimeout(int timeout)
public int getCursorLocation()
public void setCursorLocation(int cursorLoc)
public String getDefaultDatabase()
public void setDefaultDatabase(String db)
public int getIsolationLevel()
public void setIsolationLevel(int level)
public int getMode()
public void setMode(int mode)
public String getProvider()
public void setProvider(String provider)
public int getState()
public String getVersion()
public AdoProperties getProperties()
public com.ms.wfc.data.Errors getErrors()

```

Events

For more information about ADO/WFC events, see [ADO Event Instantiation by Language](#).

```

public void addOnBeginTransComplete(ConnectionEventHandler handler)
public void removeOnBeginTransComplete(ConnectionEventHandler handler)
public void addOnCommitTransComplete(ConnectionEventHandler handler)
public void removeOnCommitTransComplete(ConnectionEventHandler handler)
public void addOnConnectComplete(ConnectionEventHandler handler)
public void removeOnConnectComplete(ConnectionEventHandler handler)
public void addOnDisconnect(ConnectionEventHandler handler)
public void removeOnDisconnect(ConnectionEventHandler handler)
public void addOnExecuteComplete(ConnectionEventHandler handler)
public void removeOnExecuteComplete(ConnectionEventHandler handler)
public void addOnInfoMessage(ConnectionEventHandler handler)
public void removeOnInfoMessage(ConnectionEventHandler handler)
public void addOnRollbackTransComplete(ConnectionEventHandler handler)
public void removeOnRollbackTransComplete(ConnectionEventHandler handler)
public void addOnWillConnect(ConnectionEventHandler handler)
public void removeOnWillConnect(ConnectionEventHandler handler)
public void addOnWillExecute(ConnectionEventHandler handler)
public void removeOnWillExecute(ConnectionEventHandler handler)

```

See Also

Connection Object

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

DataSpace (ADO/WFC Syntax)

The **createObject** method of the **DataSpace** class specifies both a business object to process client application requests (*progid*) and the communications protocol and server (*connection*). **createObject** returns an [ObjectProxy](#) object that represents the server.

package com.ms.wfc.data

Constructor

```
public DataSpace()
```

Methods

```
public static ObjectProxy DataSpace.createObject(String  
    progid, String connection)
```

Properties

```
public static int getInternetTimeout()  
public static void setInternetTimeout(int pInternetTimeout)
```

See Also

[DataSpace Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Error (ADO/WFC Syntax)

package com.ms.wfc.data

Properties

```
public String getDescription\(\)  
public int getNativeError\(\)  
public int getNumber\(\)  
public String getSource\(\)  
public String getSQLState\(\)
```

See Also

[Error Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 API Reference

Field (ADO/WFC Syntax)

```
package com.ms.wfc.data
```

Methods

```
public void appendChunk(byte[] bytes)  
public void appendChunk(char[] chars)  
public void appendChunk(String chars)  
public byte[] getByteChunk(int len)  
public char[] getCharChunk(int len)  
public String getStringChunk(int len)
```

Properties

```
public int getActualSize()  
public int getAttributes()  
public void setAttributes(int pl)  
public com.ms.com.IUnknown getDataFormat()  
public void setDataFormat(com.ms.com.IUnknown format)
```

(For more information, see the Microsoft Visual J++ WFC Reference documentation for the com.ms.wfc.data.IDataFormat interface.)

```
public int getDefinedSize()  
public void setDefinedSize(int pl)  
public String getName()  
public int getNumericScale()  
public void setNumericScale(byte pbNumericScale)  
public Variant getOriginalValue()  
public int getPrecision()  
public void setPrecision(byte pbPrecision)  
public int getType()  
public void setType(int pDataType)  
public Variant getUnderlyingValue()  
public Variant getValue()  
public void setValue(Variant value)  
public AdoProperties getProperties()
```

Field Accessor Methods

The [Value](#) property of a [Field](#) object gets or sets the content of that object. The content is represented as a VARIANT, a type of object that can be assigned a

value and any of several data types.

ADO/WFC implements the **Value** property with the **getValue** method, which returns a VARIANT object; and the **setValue** method, which takes a VARIANT as an argument. VARIANTS are highly efficient in certain languages, such as Microsoft Visual Basic. However, you can attain better performance in Microsoft Visual J++ by using native Java data types.

In addition to the **Value** property, ADO/WFC provides *accessor* methods that use Java data types to get and set the content of **Field** objects. Most of these methods have names of the form **getData** or **setData**.

There are two noteworthy exceptions: One of the **getObject** methods returns an object coerced into a specified class. There is no **getNull** property; instead, there is an **isNull** property that returns a Boolean value indicating whether the field is null.

```
public native boolean getBoolean();
public void setBoolean(boolean v)
public native byte getByte();
public void setByte(byte v)
public native byte[] getBytes();
public void setBytes(byte[] v)
public native double getDouble();
public void setDouble(double v)
public native float getFloat();
public void setFloat(float v)
public native int getInt();
public void setInt(int v)
public native long getLong();
public void setLong(long v)
public native short getShort();
public void setShort(short v)
public native String getString();
public void setString(String v)
public native boolean isNull();
public void setNull()
public Object getObject()
public Object getObject(Class c)
public void setObject(Object value)
```

See Also

[Field Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

ObjectProxy (ADO/WFC Syntax)

An **ObjectProxy** object represents a server, and is returned by the **createObject** method of the [DataSpace](#) object. The ObjectProxy class has one method, **call**, which can invoke a method on the server and return an object resulting from that invocation.

```
package com.ms.wfc.data
```

Methods

Call Method (ADO/WFC Syntax)

Invokes a method on the server represented by the ObjectProxy. Optionally, method arguments may be passed as an array of objects.

Syntax

```
public Object ObjectProxy.call( String method )  
public Object ObjectProxy.call( String method, Object[] args )
```

Returns

Object

An object resulting from invoking the method.

Parameters

ObjectProxy

An **ObjectProxy** object that represents the server.

method

A String, containing the name of the method to invoke on the server.

args

Optional. An array of objects that are arguments to the method on the server. Java data types are automatically converted to data types suitable for use on the server.

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 API Reference

Parameter (ADO/WFC Syntax)

```
package com.ms.wfc.data
```

Constructor

```
public Parameter()  
public Parameter(String name)  
public Parameter(String name, int type)  
public Parameter(String name, int type, int dir)  
public Parameter(String name, int type, int dir, int size)  
public Parameter(String name, int type, int dir, int size, Object va)
```

Methods

```
public void appendChunk(byte[] bytes)  
public void appendChunk(char[] chars)  
public void appendChunk(String chars)
```

Properties

```
public int getAttributes()  
public void setAttributes(int attr)  
public int getDirection()  
public void setDirection(int dir)  
public String getName()  
public void setName(String name)  
public int getNumericScale()  
public void setNumericScale(int scale)  
public int getPrecision()  
public void setPrecision(int prec)  
public int getSize()  
public void setSize(int size)  
public int getType()  
public void setType(int type)  
public com.ms.com.Variant getValue()  
public void setValue(Object v)  
public AdoProperties getProperties()
```

Parameter Accessor Methods

The [Value](#) property of a [Parameter](#) object gets or sets the content of that object.

The content is represented as a VARIANT, a type of object that can be assigned a value and any of several data types.

ADO/WFC implements the **Value** property with the **getValue** method, which returns a VARIANT object; and the **setValue** method, which takes a VARIANT as an argument. VARIANTS are highly efficient in certain languages, such as Microsoft Visual Basic. However, you can attain better performance in Microsoft Visual J++ by using native Java data types.

In addition to the **Value** property, ADO/WFC provides *accessor* methods that use Java data types to get and set the content of **Parameter** objects. Most of these methods have names of the form **getData***Type* or **setData***Type*.

There is one noteworthy exception: There is no **getNull** property; instead, there is an **isNull** property that returns a Boolean value indicating whether the field is null.

```
public boolean getBoolean()
public void setBoolean(boolean v)
public byte getByte()
public void setByte(byte v)
public double getDouble()
public void setDouble(double v)
public float getFloat()
public void setFloat(float v)
public int getInt()
public void setInt(int v)
public long getLong()
public void setLong(long v)
public short getShort()
public void setShort(short v)
public String getString()
public void setString(String v)
public boolean isNull()
public void setNull()
```

See Also

[Parameter Object](#)

ADO 2.5 API Reference

Recordset (ADO/WFC Syntax)

package com.ms.wfc.data

Constructors

```
public Recordset()  
public Recordset(Object r)
```

Methods

```
public void addNew(Object[] fieldList, Object[] valueList)  
public void addNew(Object[] valueList)  
public void addNew()  
public void cancel()  
public void cancelBatch(int affectRecords)  
public void cancelBatch()  
public void cancelUpdate()  
public Object clone()  
public Object clone(int lockType)  
public void close()  
public int compareBookmarks(Object bookmark1, Object bookmark2)  
public void delete(int affectRecords)  
public void delete()  
public void find(String criteria)  
public void find(String criteria, int SkipRecords)  
public void find(String criteria, int SkipRecords, int searchDirecti  
public void find(String criteria, int SkipRecords, int searchDirecti  
public Object[][] getRows(int Rows, Object bmkStart, Object[] fieldL  
public void move(int numRecords)  
public void move(int numRecords, Object bmkStart)  
public void moveFirst()  
public void moveLast()  
public void moveNext()  
public void movePrevious()  
public Recordset nextRecordset()  
public Recordset nextRecordset(int[] recordsAffected)  
public void open()  
public void open(Object source)  
public void open(Object source, Object activeConnection)  
public void open(Object source, Object activeConnection, int cursorT  
public void open(Object source, Object activeConnection, int cursorT  
int lockType)  
public void open(Object source, Object activeConnection, int cursorT  
int lockType, int options)
```

```
public void requery()
public void requery(int options)
public void resync()
public void resync(int affectRecords, int resyncValues)
public void save(String fileName)
public void save(String fileName, int persistFormat)
public boolean supports(int cursorOptions)
public void update()
public void update(Object[] valueList)
public void update(Object[] fieldList, Object[] valueList)
public void updateBatch()
public void updateBatch(int affectRecords)
```

Properties

```
public int getAbsolutePage()
public void setAbsolutePage(int page)
public int getAbsolutePosition()
public void setAbsolutePosition(int pos)
public Command getActiveCommand()
public Connection getActiveConnection()
public void setActiveConnection(String conn)
public void setActiveConnection(com.ms.wfc.data.Connection c)
public boolean getBOF()
public boolean getEOF()
public Object getBookmark()
public void setBookmark(Object bm)
public int getCacheSize()
public void setCacheSize(int size)
public int getCursorLocation()
public void setCursorLocation(int cursorLoc)
public int getCursorType()
public void setCursorType(int cursorType)
public String getDataMember()
public void setDataMember(String pbstrDataMember)
public IUnknown getDataSource()
public void setDataSource(IUnknown dataSource)
public int getEditMode()
public Object getFilter()
public void setFilter(Object filter)
public int getLockType()
public void setLockType(int lockType)
public int getMarshalOptions()
public void setMarshalOptions(int options)
public int getMaxRecords()
public void setMaxRecords(int maxRecords)
public int getPageCount()
public int getPageSize()
public void setPageSize(int pageSize)
```

```
public int getRecordCount()
public String getSort()
public void setSort(String criteria)
public String getSource()
public void setSource(String query)
public void setSource(com.ms.wfc.data.Command command)
public int getState()
public int getStatus()
public boolean getStayInSync()
public void setStayInSync(boolean pbStayInSync)
public com.ms.wfc.data.Field getField(int n)
public com.ms.wfc.data.Field getField(String n)
public com.ms.wfc.data.Fields getFields()
public AdoProperties getProperties()
```

Events

For more information about ADO/WFC events, see [ADO Event Instantiation by Language](#).

```
public void addOnEndOfRecordset(RecordsetEventHandler handler)
public void removeOnEndOfRecordset(RecordsetEventHandler handler)
public void addOnFetchComplete(RecordsetEventHandler handler)
public void removeOnFetchComplete(RecordsetEventHandler handler)
public void addOnFetchProgress(RecordsetEventHandler handler)
public void removeOnFetchProgress(RecordsetEventHandler handler)
public void addOnFieldChangeComplete(RecordsetEventHandler handler)
public void removeOnFieldChangeComplete(RecordsetEventHandler handler)
public void addOnMoveComplete(RecordsetEventHandler handler)
public void removeOnMoveComplete(RecordsetEventHandler handler)
public void addOnRecordChangeComplete(RecordsetEventHandler handler)
public void removeOnRecordChangeComplete(RecordsetEventHandler handler)
public void addOnRecordsetChangeComplete(RecordsetEventHandler handler)
public void removeOnRecordsetChangeComplete(RecordsetEventHandler handler)
public void addOnWillChangeField(RecordsetEventHandler handler)
public void removeOnWillChangeField(RecordsetEventHandler handler)
public void addOnWillChangeRecord(RecordsetEventHandler handler)
public void removeOnWillChangeRecord(RecordsetEventHandler handler)
public void addOnWillChangeRecordset(RecordsetEventHandler handler)
public void removeOnWillChangeRecordset(RecordsetEventHandler handler)
public void addOnWillMove(RecordsetEventHandler handler)
public void removeOnWillMove(RecordsetEventHandler handler)
```

See Also

[Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

ADO Code Examples

Use the following code examples to learn how to use the ADO objects, methods, properties, and events.

Note Paste the entire code example into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

- [ADO Code Examples in Microsoft Visual Basic](#)
- [ADO Code Examples in Microsoft Visual Basic Scripting Edition](#)
- [ADO Code Examples in Microsoft Visual C++](#)
- [ADO Code Examples in Microsoft Visual J++](#)
- [ADO Code Examples in Microsoft JScript](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ADO Code Examples in Microsoft Visual Basic

Use the following code examples to learn how to use the ADO methods, properties, and events when writing in Visual Basic.

Note Paste the entire code example, from Sub to End Sub, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

Methods

- [AddNew Method Example](#)
- [Append and CreateParameter Methods Example](#)
- [AppendChunk and GetChunk Methods Example](#)
- [BeginTrans, CommitTrans, and RollbackTrans Methods Example](#)
- [Cancel Method Example](#)
- [Clone Method Example](#)
- [CompareBookmarks Method Example](#)
- [ConvertToString Method Example](#)
- [CopyRecord, CopyTo, and SaveToFile Methods Example](#)
- [CreateRecordset Method Example](#)
- [Delete Method Example](#)
- [DeleteRecord and MoveRecord Methods Example](#)
- [Execute, Requery, and Clear Methods Example](#)
- [Find Method Example](#)
- [GetRows Method Example](#)
- [GetString Method Example](#)
- [SkipLine Method, EOS, and LineSeparator Properties Example](#)
- [Move Method Example](#)
- [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods Example](#)
- [NextRecordset Method Example](#)
- [Open and Close Methods Example](#)
- [OpenSchema Method Example](#)
- [Read, ReadText, Write, and WriteText Methods Example](#)
- [Refresh Method Example](#)

- [Resync Method Example](#)
- [Save and Open Methods Example](#)
- [Seek Method and Index Property Example](#)
- [Supports Method Example](#)
- [Update and CancelUpdate Methods Example](#)
- [UpdateBatch and CancelBatch Methods Example](#)

Properties

- [AbsolutePage, PageCount, and PageSize Properties Example](#)
- [AbsolutePosition and CursorLocation Properties Example](#)
- [ActiveCommand Property Example](#)
- [ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example](#)
- [ActualSize and DefinedSize Properties Example](#)
- [Attributes and Name Properties Example](#)
- [BOF, EOF, and Bookmark Properties Example](#)
- [CacheSize Property Example](#)
- [ConnectionString, ConnectionTimeout, and State Properties Example](#)
- [Count Property Example](#)
- [CursorType, LockType, and EditMode Properties Example](#)
- [Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState Properties Example](#)
- [EOS and LineSeparator Properties, SkipLine Method Example](#)
- [Filter and RecordCount Properties Example](#)
- [IsolationLevel and Mode Properties Example](#)
- [Item Property Example](#)
- [MarshalOptions Property Example](#)
- [MaxRecords Property Example](#)
- [NumericScale and Precision Properties Example](#)
- [Optimize Property Example](#)
- [OriginalValue and UnderlyingValue Properties Example](#)
- [Prepared Property Example](#)
- [Provider and DefaultDatabase Properties Example](#)
- [Sort Property Example](#)
- [Source Property Example](#)
- [State Property Example](#)
- [Status Property Example](#)
- [StayInSync Property Example](#)

- [Type Property Example \(Field\)](#)
- [Type Property Example \(Property\)](#)
- [Value Property Example](#)
- [Version Property Example](#)

See Also

[ADO Code Examples in Microsoft Visual Basic Scripting Edition](#) | [ADO Code Examples in Microsoft Visual C++](#) | [ADO Code Examples in Microsoft Visual J++](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AbsolutePage, PageCount, and PageSize Properties Example (VB)

```
'BeginAbsolutePageVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQL As String
    'record variables
    Dim strMessage As String
    Dim intPage As Integer
    Dim intPageCount As Integer
    Dim intRecord As Integer

    'Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open employee recordset
    ' Use client cursor to enable AbsolutePosition property
    Set rstEmployees = New ADODB.Recordset
    strSQL = "employee"
    rstEmployees.Open strSQL, strCnxn, adUseClient, adLockReadOnly,

    ' Display names and hire dates, five records at a time
    rstEmployees.PageSize = 5
    intPageCount = rstEmployees.PageCount
    For intPage = 1 To intPageCount
        rstEmployees.AbsolutePage = intPage
        strMessage = ""
        For intRecord = 1 To rstEmployees.PageSize
            strMessage = strMessage & _
                rstEmployees!fname & " " & _
```

```

        rstEmployees!lname & " " & _
        rstEmployees!hire_date & vbCr
    rstEmployees.MoveNext
    If rstEmployees.EOF Then Exit For
Next intRecord
MsgBox strMessage
Next intPage

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

```

ErrorHandler:

```

' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndAbsolutePageVB

```

See Also

[AbsolutePage Property](#) | [PageCount Property](#) | [PageSize Property](#) | [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

AbsolutePosition and CursorLocation Properties Example (VB)

This example demonstrates how the [AbsolutePosition](#) property can track the progress of a loop that enumerates all the records of a [Recordset](#). It uses the [CursorLocation](#) property to enable the **AbsolutePosition** property by setting the [cursor](#) to a [client](#) cursor.

```
'BeginAbsolutePositionVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQL As String
        'record variables
    Dim strMessage As String

    'Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open Employee recordset with
    ' Client-side cursor to enable AbsolutePosition property
    Set rstEmployees = New ADODB.Recordset
    strSQL = "employee"
    rstEmployees.Open strSQL, strCnxn, adUseClient, adLockReadOnly,

    ' Enumerate Recordset
    Do While Not rstEmployees.EOF
        ' Display current record information
        strMessage = "Employee: " & rstEmployees!lname & vbCr & _
            "(record " & rstEmployees.AbsolutePosition & _
            " of " & rstEmployees.RecordCount & ")"
```

```

        If MsgBox(strMessage, vbOKCancel) = vbCancel Then Exit Do
        rstEmployees.MoveNext
Loop

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndAbsolutePositionVB

```

See Also

[AbsolutePosition Property](#) | [CursorLocation Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActiveCommand Property Example (VB)

This example demonstrates the [ActiveCommand](#) property.

A subroutine is given a [Recordset](#) object whose **ActiveCommand** property is used to display the command text and parameter that created the **Recordset**.

```
'BeginActiveCommandVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

        'recordset and connection variables
    Dim cmd As ADODB.Command
    Dim rst As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
        'record variables
    Dim strPrompt As String
    Dim strName As String

    Set Cnxn = New ADODB.Connection
    Set cmd = New ADODB.Command

    strPrompt = "Enter an author's name (e.g., Ringer): "
    strName = Trim(InputBox(strPrompt, "ActiveCommandX Example"))
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"

        'create SQL command string
    cmd.CommandText = "SELECT * FROM Authors WHERE au_lname = ?"
    cmd.Parameters.Append cmd.CreateParameter("LastName", adChar, ad

    Cnxn.Open strCnxn
    cmd.ActiveConnection = Cnxn

        'create the recordset by executing command string
    Set rst = cmd.Execute(, , adCmdText)
```

```

        'see the results
    Call ActiveCommandXprint(rst)

    ' clean up
    Cnxn.Close
    Set rst = Nothing
    Set Cnxn = Nothing
    Exit Sub

ErrorHandler:
    ' clean up
    If Not rst Is Nothing Then
        If rst.State = adStateOpen Then rst.Close
    End If
    Set rst = Nothing

    If Not Cnxn Is Nothing Then
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
    Set Cnxn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndActiveCommandVB

```

The **ActiveCommandXprint** routine is given only a **Recordset** object, yet it must print the command text and parameter that created the **Recordset**. This can be done because the **Recordset** object's **ActiveCommand** property yields the associated [Command](#) object.

The **Command** object's [CommandText](#) property yields the parameterized command that created the **Recordset**. The **Command** object's [Parameters](#) collection yields the value that was substituted for the command's parameter placeholder ("?").

Finally, an error message or the author's name and ID are printed.

```

'BeginActiveCommandPrintVB
Public Sub ActiveCommandXprint(rstp As ADODB.Recordset)

    Dim strName As String

    strName = rstp.ActiveCommand.Parameters.Item("LastName").Value

    Debug.Print "Command text = "; rstp.ActiveCommand.CommandText;

```

```
Debug.Print "Parameter = "; strName; ""

If rstp.BOF = True Then
    Debug.Print "Name = "; strName; ', not found.'
Else
    Debug.Print "Name = "; rstp!au_fname; " "; rstp!au_lname; _
        "', author ID = "; rstp!au_id; ""
End If

rstp.Close
Set rstp = Nothing
End Sub
```

'EndActiveCommandPrintVBSee Also

[ActiveCommand Property](#) | [Command Object](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example (VB)

This example uses the [ActiveConnection](#), [CommandText](#), [CommandTimeout](#), [CommandType](#), [Size](#), and [Direction](#) properties to execute a stored procedure.

```
'BeginActiveConnectionVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset, command and connection variables
    Dim Cnxn As ADODB.Connection
    Dim cmdByRoyalty As ADODB.Command
    Dim prmByRoyalty As ADODB.Parameter
    Dim rstByRoyalty As ADODB.Recordset
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String
    Dim strSQLByRoyalty As String
    'record variables
    Dim intRoyalty As Integer
    Dim strAuthorID As String

    ' Define a command object for a stored procedure
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    Set cmdByRoyalty = New ADODB.Command
    Set cmdByRoyalty.ActiveConnection = Cnxn
    ' Set the criteria
    strSQLByRoyalty = "byroyalty"
```

```

cmdByRoyalty.CommandText = strSQLByRoyalty
cmdByRoyalty.CommandType = adCmdStoredProc
cmdByRoyalty.CommandTimeout = 15

' Define the stored procedure's input parameter
intRoyalty = Trim(InputBox("Enter royalty:"))
Set prmByRoyalty = New ADODB.Parameter
prmByRoyalty.Type = adInteger
prmByRoyalty.Size = 3
prmByRoyalty.Direction = adParamInput
prmByRoyalty.Value = intRoyalty

cmdByRoyalty.Parameters.Append prmByRoyalty

' Create a recordset by executing the command.
Set rstByRoyalty = cmdByRoyalty.Execute()

' Open the Authors Table to get author names for display
Set rstAuthors = New ADODB.Recordset
strSQLAuthors = "Authors"

'rstAuthors.Open strSQLAuthors, strCnxn, , , adCmdTable
rstAuthors.Open strSQLAuthors, strCnxn, adOpenForwardOnly, adLoc
'the above two lines of code are identical as the default values
'CursorType and LockType arguments match those shown

' Print the recordset and add author names from Table
Debug.Print "Authors with " & intRoyalty & _
    " percent royalty"

Do Until rstByRoyalty.EOF
    strAuthorID = rstByRoyalty!au_id
    Debug.Print , rstByRoyalty!au_id & ", ";
    rstAuthors.Filter = "au_id = '" & strAuthorID & "'"
    Debug.Print rstAuthors!au_fname & " " & _
        rstAuthors!au_lname
    rstByRoyalty.MoveNext
Loop

' clean up
rstAuthors.Close
rstByRoyalty.Close
Cnxn.Close
Set rstAuthors = Nothing
Set rstByRoyalty = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up

```

```
If Not rstAuthors Is Nothing Then
    If rstAuthors.State = adStateOpen Then rstAuthors.Close
End If
Set rstAuthors = Nothing

If Not rstByRoyalty Is Nothing Then
    If rstByRoyalty.State = adStateOpen Then rstByRoyalty.Close
End If
Set rstByRoyalty = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndActiveConnectionVB
```

See Also

[ActiveCommand Property](#) | [Command Object](#) | [CommandText Property](#) | [CommandTimeout Property](#) | [CommandType Property](#) | [Connection Object](#) | [Direction Property](#) | [Parameter Object](#) | [Record Object](#) | [Recordset Object](#) | [Size Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActualSize and DefinedSize Properties Example (VB)

This example uses the [ActualSize](#) and [DefinedSize](#) properties to display the defined size and actual size of a field.

```
'BeginActualSizeVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim rstStores As ADODB.Recordset
    Dim SQLStores As String
    Dim strCnxn As String
    'record variables
    Dim strMessage As String

    ' Open a recordset for the Stores table
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Northwind';Integrated Security='SSPI';"
    Set rstStores = New ADODB.Recordset

    SQLStores = "Suppliers"
    rstStores.Open SQLStores, strCnxn, adOpenForwardOnly, adLockRead
    'the above two lines of code are identical as the default values
    'CursorType and LockType arguments match those indicated

    ' Loop through the recordset displaying the contents
    ' of the store_name field, the field's defined size,
    ' and its actual size.
    rstStores.MoveFirst

    Do Until rstStores.EOF
        strMessage = "Company name: " & rstStores!CompanyName & _
            vbCrLf & "Defined size: " & _
            rstStores!CompanyName.DefinedSize & _
            vbCrLf & "Actual size: " & _
            rstStores!CompanyName.ActualSize & vbCrLf
```

```
        MsgBox strMessage, vbOKCancel, "ADO ActualSize Property (Vis
        rstStores.MoveNext
Loop

    ' clean up
    rstStores.Close
    Set rstStores = Nothing
    Exit Sub

ErrorHandler:
    ' clean up
    If Not rstStores Is Nothing Then
        If rstStores.State = adStateOpen Then rstStores.Close
    End If
    Set rstStores = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndActualSizeVB
```

See Also

[ActualSize Property](#) | [DefinedSize Property](#) | [Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AddNew Method Example (VB)

This example uses the [AddNew](#) method to create a new record with the specified name.

```
'BeginAddNewVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim rstEmployees As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQL As String
    'record variables
    Dim strID As String
    Dim strFirstName As String
    Dim strLastName As String
    Dim blnRecordAdded As Boolean

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Northwind';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open Employees Table with a cursor that allows updates
    Set rstEmployees = New ADODB.Recordset
    strSQL = "Employees"
    rstEmployees.Open strSQL, strCnxn, adOpenKeyset, adLockOptimisti

    ' Get data from the user
    strFirstName = Trim(InputBox("Enter first name:"))
    strLastName = Trim(InputBox("Enter last name:"))

    ' Proceed only if the user actually entered something
    ' for both the first and last names
    If strFirstName <> "" And strLastName <> "" Then

        rstEmployees.AddNew
```

```

rstEmployees!firstname = strFirstName
rstEmployees!LastName = strLastName
rstEmployees.Update
blnRecordAdded = True

' Show the newly added data
MsgBox "New record: " & rstEmployees!EmployeeId & " " & _
rstEmployees!firstname & " " & rstEmployees!LastName

Else
    MsgBox "Please enter a first name and last name."
End If

' Delete the new record because this is a demonstration
Cnxn.Execute "DELETE FROM Employees WHERE EmployeeID = '" & strI

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndAddNewVB

```

See Also

[AddNew Method](#) | [Recordset Object](#)

ADO 2.5 Samples 

Append and CreateParameter Methods Example (VB)

This example uses the [Append](#) and [CreateParameter](#) methods to execute a stored procedure with an input parameter.

```
'BeginAppendVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset, command and connection variables
    Dim Cnxn As ADODB.Connection
    Dim cmdByRoyalty As ADODB.Command
    Dim prmByRoyalty As ADODB.Parameter
    Dim rstByRoyalty As ADODB.Recordset
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String
    Dim strSQLByRoyalty As String
    'record variables
    Dim intRoyalty As Integer
    Dim strAuthorID As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open command object with one parameter
    Set cmdByRoyalty = New ADODB.Command
    cmdByRoyalty.CommandText = "byroyalty"
    cmdByRoyalty.CommandType = adCmdStoredProc

    ' Get parameter value and append parameter
    intRoyalty = Trim(InputBox("Enter royalty:"))
    Set prmByRoyalty = cmdByRoyalty.CreateParameter("percentage", ad
    cmdByRoyalty.Parameters.Append prmByRoyalty
    prmByRoyalty.Value = intRoyalty
```

```

' Create recordset by executing the command
Set cmdByRoyalty.ActiveConnection = Cnxn
Set rstByRoyalty = cmdByRoyalty.Execute

' Open the Authors Table to get author names for display
' and set cursor client-side
Set rstAuthors = New ADODB.Recordset
strSQLAuthors = "Authors"
rstAuthors.Open strSQLAuthors, Cnxn, adUseClient, adLockOptimist

' Print recordset adding author names from Authors table
Debug.Print "Authors with " & intRoyalty & " percent royalty"

Do Until rstByRoyalty.EOF
    strAuthorID = rstByRoyalty!au_id
    Debug.Print "    " & rstByRoyalty!au_id & ", ";
    rstAuthors.Filter = "au_id = '" & strAuthorID & "'"
    Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname
    rstByRoyalty.MoveNext
Loop

' clean up
rstByRoyalty.Close
rstAuthors.Close
Cnxn.Close
Set rstByRoyalty = Nothing
Set rstAuthors = Nothing
Set Cnxn = Nothing
Exit Sub

```

ErrorHandler:

```

' clean up
If Not rstByRoyalty Is Nothing Then
    If rstByRoyalty.State = adStateOpen Then rstByRoyalty.Close
End If
Set rstByRoyalty = Nothing

If Not rstAuthors Is Nothing Then
    If rstAuthors.State = adStateOpen Then rstAuthors.Close
End If
Set rstAuthors = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"

```

```
End If  
End Sub  
'EndAppendVB
```

See Also

[Append Method](#) | [CreateParameter Method](#) | [Field Object](#) | [Fields Collection](#) | [Parameter Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AppendChunk and GetChunk Methods Example (VB)

This example uses the [AppendChunk](#) and [GetChunk](#) methods to fill an image field with data from another record.

```
'BeginAppendChunkVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim rstPubInfo As ADODB.Recordset
    Dim strSQLPubInfo As String
    'record variables
    Dim strPubID As String
    Dim strPRInfo As String
    Dim lngOffset As Long
    Dim lngLogoSize As Long
    Dim varLogo As Variant
    Dim varChunk As Variant
    Dim strMsg As String

    Const conChunkSize = 100

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open the pub_info table with a cursor that allows updates
    Set rstPubInfo = New ADODB.Recordset
    strSQLPubInfo = "pub_info"
    rstPubInfo.Open strSQLPubInfo, Cnxn, adOpenKeyset, adLockOptimis

    ' 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, ",")
            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" & _
    " [must be > 9899 & < 9999]:"))

strPRInfo = Trim(InputBox("Enter descriptive text:"))

' Add the new publisher to the publishers table to avoid
' getting an error due to foreign key constraint
Cnxn.Execute "INSERT publishers(pub_id, pub_name) VALUES('" & _
    strPubID & "', 'Your Test Publisher'"

' 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

' Delete new records because this is a demo

```

```

rstPubInfo.Requery
Cnxn.Execute "DELETE FROM pub_info " & _
    "WHERE pub_id = '" & strPubID & "'"

Cnxn.Execute "DELETE FROM publishers " & _
    "WHERE pub_id = '" & strPubID & "'"

' clean up
rstPubInfo.Close
Cnxn.Close
Set rstPubInfo = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstPubInfo Is Nothing Then
    If rstPubInfo.State = adStateOpen Then rstPubInfo.Close
End If
Set rstPubInfo = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndAppendChunkVB

```

See Also

[AppendChunk Method](#) | [Field Object](#) | [GetChunk Method](#) | [Parameter Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Attributes and Name Properties

Example (VB)

This example displays the value of the [Attributes](#) property for [Connection](#), [Field](#), and [Property](#) objects. It uses the [Name](#) property to display the name of each **Field** and **Property** object.

```
'BeginAttributesVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim rstEmployees As ADODB.Recordset
    Dim strSQLEmployee As String
    'record variables
    Dim adoField As ADODB.Field
    Dim adoProp As ADODB.Property

    ' Open connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Open recordset
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployee = "employee"
    rstEmployees.Open strSQLEmployee, Cnxn, adOpenForwardOnly, adLoc
    'the above two lines openign the recordset are identical as
    'the default values for CursorType and LockType arguments match

    ' Display the attributes of the connection
    Debug.Print "Connection attributes = " & Cnxn.Attributes

    ' Display the property attributes of the Employee Table
    Debug.Print "Property attributes:"
```

```

For Each adoProp In rstEmployees.Properties
    Debug.Print "    " & adoProp.Name & " = " & adoProp.Attribute
Next adoProp

' Display the field attributes of the Employee Table
Debug.Print "Field attributes:"
For Each adoField In rstEmployees.Fields
    Debug.Print "    " & adoField.Name & " = " & adoField.Attribute
Next adoField

' Display fields of the Employee Table which are NULLABLE
Debug.Print "NULLABLE Fields:"
For Each adoField In rstEmployees.Fields
    If CBool(adoField.Attributes And adFldIsNullable) Then
        Debug.Print "    " & adoField.Name
    End If
Next adoField

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

```

ErrorHandler:

```

' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndAttributesVB

```

See Also

[Attributes Property](#) | [Connection Object](#) | [Field Object](#) | [Name Property](#) | [Property Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

BeginTrans, CommitTrans, and RollbackTrans Methods Example (VB)

This example changes the book type of all psychology books in the *Titles* table of the database. After the [BeginTrans](#) method starts a transaction that isolates all the changes made to the *Titles* table, the [CommitTrans](#) method saves the changes. You can use the [RollbackTrans](#) method to undo changes that you saved using the [Update](#) method.

```
'BeginBeginTransVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim rstTitles As ADODB.Recordset
    Dim strSQLTitles As String
    'record variables
    Dim strTitle As String
    Dim strMessage As String

    ' Open connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Open recordset dynamic to allow for changes
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "Titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenDynamic, adLockPessimis

    Cnxn.BeginTrans
```

```

' Loop through recordset and prompt user
' to change the type for a specified title

rstTitles.MoveFirst

Do Until rstTitles.EOF
    If Trim(rstTitles!Type) = "psychology" Then
        strTitle = rstTitles!Title
        strMessage = "Title: " & strTitle & vbCr & _
            "Change type to self help?"

        ' If yes, change type for the specified title
        If MsgBox(strMessage, vbYesNo) = vbYes Then
            rstTitles!Type = "self_help"
            rstTitles.Update
        End If
    End If
    rstTitles.MoveNext
Loop

' Prompt user to commit all changes made
If MsgBox("Save all changes?", vbYesNo) = vbYes Then
    Cnxn.CommitTrans
Else
    Cnxn.RollbackTrans
End If

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

' Restore original data as this is a demo
rstTitles.MoveFirst

Do Until rstTitles.EOF
    If Trim(rstTitles!Type) = "self_help" Then
        rstTitles!Type = "psychology"
        rstTitles.Update
    End If
    rstTitles.MoveNext
Loop

' clean up
rstTitles.Close
Cnxn.Close
Set rstTitles = Nothing

```

```
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then rstTitles.Close
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub

'EndBeginTransVB
```

See Also

[BeginTrans, CommitTrans, and RollbackTrans Methods](#) | [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

BOF, EOF, and Bookmark Properties

Example (VB)

This example uses the [BOF](#) and [EOF](#) properties to display a message if a user tries to move past the first or last record of a [Recordset](#). It uses the [Bookmark](#) property to let the user flag a record in a **Recordset** and return to it later.

```
'BeginBOFVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim rstPublishers As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLPubs As String
    'record variables
    Dim strMessage As String
    Dim intCommand As Integer
    Dim varBookmark As Variant

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset and use client cursor
    ' to enable AbsolutePosition property
    Set rstPublishers = New ADODB.Recordset
    strSQLPubs = "SELECT pub_id, pub_name FROM publishers ORDER BY p
rstPublishers.Open strSQLPubs, strCnxn, adUseClient, adOpenStati

rstPublishers.MoveFirst
Do Until rstPublishers.EOF
    ' Display information about current record
    ' and get user input
    strMessage = "Publisher: " & rstPublishers!pub_name & _
```

```

        vbCr & "(record " & rstPublishers.AbsolutePosition & _
        " of " & rstPublishers.RecordCount & ")" & vbCr & vbCr &
        "Enter command:" & vbCr & _
        "[1 - next / 2 - previous /" & vbCr & _
        "3 - set bookmark / 4 - go to bookmark]"
intCommand = Val(InputBox(strMessage))

' Check user input
Select Case intCommand
    Case 1
        ' Move forward trapping for EOF
        rstPublishers.MoveNext
        If rstPublishers.EOF Then
            MsgBox "Moving past the last record." & _
                vbCr & "Try again."
            rstPublishers.MoveLast
        End If
    Case 2
        ' Move backward trapping for BOF
        rstPublishers.MovePrevious
        If rstPublishers.BOF Then
            MsgBox "Moving past the first record." & _
                vbCr & "Try again."
            rstPublishers.MoveFirst
        End If
    Case 3
        ' Store the bookmark of the current record
        varBookmark = rstPublishers.Bookmark
    Case 4
        ' Go to the record indicated by the stored bookmark
        If IsEmpty(varBookmark) Then
            MsgBox "No Bookmark set!"
        Else
            rstPublishers.Bookmark = varBookmark
        End If
    Case Else
        Exit Do
End Select
Loop

' clean up
rstPublishers.Close
Cnxn.Close
Set rstPublishers = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstPublishers Is Nothing Then

```

```

        If rstPublishers.State = adStateOpen Then rstPublishers.Clos
End If
Set rstPublishers = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndBOFVB

```

This example uses the **Bookmark** and [Filter](#) properties to create a limited view of the **Recordset**. Only records referenced by the array of bookmarks are accessible.

```

'BeginBOF2VB
Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim rs As New ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQL As String
    Dim strCnxn As String

    Dim bmk(10)

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySQLServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    'open the recordset client-side
    Set rs = New ADODB.Recordset
    strSQL = "Select * from Authors"
    rs.Open strSQL, Cnxn, adUseClient, adLockReadOnly, adCmdText
    Debug.Print "Number of records before filtering: ", rs.RecordCou

    Dim ii As Integer
    ii = 0

    If rs.EOF <> True And ii < 11 Then
        Do

```

```

        If Not (rs.EOF <> True And ii < 11) Then Exit Do
        bmk(ii) = rs.Bookmark
        ii = ii + 1
        rs.Move 2
    Loop Until rs.EOF
End If

rs.Filter = bmk
Debug.Print "Number of records after filtering: ", rs.RecordCount

rs.MoveFirst
If rs.EOF <> True Then
    Do
        Debug.Print rs.AbsolutePosition, rs("au_lname")
        rs.MoveNext
    Loop Until rs.EOF
End If

' clean up
rs.Close
Cnxn.Close
Set rs = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rs Is Nothing Then
    If rs.State = adStateOpen Then rs.Close
End If
Set rs = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndB0F2VB

```

See Also

[BOF, EOF Properties](#) | [Bookmark Property](#) | [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

CacheSize Property Example (VB)

This example uses the [CacheSize](#) property to show the difference in performance for an operation performed with and without a 30-record cache.

```
'BeginCacheSizeVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim rstRoySched As ADODB.Recordset
    Dim strSQLSched As String
    Dim strCnxn As String
    'record variables
    Dim sngStart As Single
    Dim sngEnd As Single
    Dim sngNoCache As Single
    Dim sngCache As Single
    Dim intLoop As Integer
    Dim strTemp As String

    ' Open the connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"

    ' Open the RoySched Table
    Set rstRoySched = New ADODB.Recordset
    strSQLSched = "roysched"
    rstRoySched.Open strSQLSched, strCnxn, , , adCmdTable

    ' Enumerate the Recordset object twice and
    ' record the elapsed time
    sngStart = Timer

    For intLoop = 1 To 2
        rstRoySched.MoveFirst

        If Not rstRoySched.EOF Then
            ' Execute a simple operation for the
            ' performance test
```

```

        Do
            strTemp = rstRoySched!title_id
            rstRoySched.MoveNext
        Loop Until rstRoySched.EOF
    End If
Next intLoop

sngEnd = Timer
sngNoCache = sngEnd - sngStart

' Cache records in groups of 30 records.
rstRoySched.MoveFirst
rstRoySched.CacheSize = 30
sngStart = Timer

' Enumerate the Recordset object twice and record
' the elapsed time
For intLoop = 1 To 2
    rstRoySched.MoveFirst
    Do While Not rstRoySched.EOF
        ' Execute a simple operation for the
        ' performance test
        strTemp = rstRoySched!title_id
        rstRoySched.MoveNext
    Loop
Next intLoop

sngEnd = Timer
sngCache = sngEnd - sngStart

' Display performance results.
MsgBox "Caching Performance Results:" & vbCrLf & _
    "    No cache: " & Format(sngNoCache, "##0.000") & " seconds"
    "    30-record cache: " & Format(sngCache, "##0.000") & " sec

' clean up
rstRoySched.Close
Set rstRoySched = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstRoySched Is Nothing Then
    If rstRoySched.State = adStateOpen Then rstRoySched.Close
End If
Set rstRoySched = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

```

End Sub
'EndCacheSizeVB

See Also

[CacheSize Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Cancel Method Example (VB)

This example uses the [Cancel](#) method to cancel a command executing on a [Connection](#) object if the connection is busy.

```
'BeginCancelVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strCmdChange As String
    Dim strCmdRestore As String
    'record variables
    Dim blnChanged As Boolean

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Define command strings
    strCmdChange = "UPDATE titles SET type = 'self_help' WHERE type
    strCmdRestore = "UPDATE titles SET type = 'psychology' " & _
        "WHERE type = 'self_help'"

    ' Begin a transaction, then execute a command asynchronously
    Cnxn.BeginTrans
    Cnxn.Execute strCmdChange, , adAsyncExecute
    ' do something else for a little while -
    ' use i = 1 to 32000 to allow completion
    Dim i As Integer
    For i = 1 To 1000
        i = i + i
        Debug.Print i
    Next i

    ' If the command has NOT completed, cancel the execute and
```

```

' roll back the transaction; otherwise, commit the transaction
If CBool(Cnxn.State And adStateExecuting) Then
    Cnxn.Cancel
    Cnxn.RollbackTrans
    blnChanged = False
    MsgBox "Update canceled."
Else
    Cnxn.CommitTrans
    blnChanged = True
    MsgBox "Update complete."
End If

' If the change was made, restore the data
' because this is only a demo
If blnChanged Then
    Cnxn.Execute strCmdRestore
    MsgBox "Data restored."
End If

' clean up
Cnxn.Close
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndCancelVB

```

See Also

[Cancel Method](#) | [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Clone Method Example (VB)

This example uses the [Clone](#) method to create copies of a [Recordset](#) and then lets the user position the record pointer of each copy independently.

```
'BeginCloneVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset array and connection variables
    Dim arstStores(1 To 3) As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLStore As String
    Dim strCnxn As String
    'record variables
    Dim intLoop As Integer
    Dim strMessage As String
    Dim strFind As String

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset as a static cursor type recordset
    Set arstStores(1) = New ADODB.Recordset
    strSQLStore = "SELECT stor_name FROM Stores ORDER BY stor_name"
    arstStores(1).Open strSQLStore, strCnxn, adOpenStatic, adLockBat

    ' Create two clones of the original Recordset
    Set arstStores(2) = arstStores(1).Clone
    Set arstStores(3) = arstStores(1).Clone

    ' Loop through the array so that on each pass the user
    ' is searching a different copy of the same Recordset
    Do
        For intLoop = 1 To 3
            ' Ask for search string while showing where
            ' the current record pointer is for each Recordset
```

```

strMessage = _
    "Recordsets from stores table:" & vbCr & _
    "  1 - Original - Record pointer at " & arstStores(1)
    "  2 - Clone - Record pointer at " & arstStores(2)!s
    "  3 - Clone - Record pointer at " & arstStores(3)!s
    "Enter search string for #" & intLoop & ":"

strFind = Trim(InputBox(strMessage))
' make sure something was entered, if not then EXIT loc
If strFind = "" Then Exit Do

' otherwise locate the record from the entered string
arstStores(intLoop).Filter = "stor_name = '" & strFind &

'if there's no match, jump to the last record
If arstStores(intLoop).EOF Then
    arstStores(intLoop).Filter = adFilterNone
    arstStores(intLoop).MoveLast
Else
    MsgBox "Found " & strFind
End If
Next intLoop
Loop

' clean up
arstStores(1).Close
arstStores(2).Close
arstStores(3).Close
Cnxn.Close
Set arstStores(1) = Nothing
Set arstStores(2) = Nothing
Set arstStores(3) = Nothing
Set Cnxn = Nothing
Exit Sub

```

ErrorHandler:

```

' clean up
If Not arstStores(1) Is Nothing Then
    If arstStores(1).State = adStateOpen Then arstStores(1).Close
End If
Set arstStores(1) = Nothing
If Not arstStores(2) Is Nothing Then
    If arstStores(2).State = adStateOpen Then arstStores(2).Close
End If
Set arstStores(2) = Nothing
If Not arstStores(3) Is Nothing Then
    If arstStores(3).State = adStateOpen Then arstStores(3).Close
End If
Set arstStores(3) = Nothing

```

```
If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndCloneVB
```

See Also

[Clone Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

CompareBookmarks Method Example (VB)

This example demonstrates the [CompareBookmarks](#) method. The relative value of bookmarks is seldom needed unless a particular bookmark is somehow special.

Designate a random row of a [Recordset](#) derived from the *Authors* table as the target of a search. Then display the position of each row relative to that target.

```
'BeginCompareBookmarksVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' recordset and connection variables
    Dim rstAuthors As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLAuthors As String
    Dim strCnxn As String

    ' comparison variables
    Dim count As Integer
    Dim target As Variant
    Dim result As Long
    Dim strAnswer As String
    Dim strTitle As String
    strTitle = "CompareBookmarks Example"

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset as a static cursor type recordset
    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "SELECT * FROM Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnl
```

```

count = rstAuthors.RecordCount
Debug.Print "Rows in the Recordset = "; count

' Exit if an empty recordset
If count = 0 Then Exit Sub

' Get position between 0 and count -1
Randomize
count = (Int(count * Rnd))
Debug.Print "Randomly chosen row position = "; count
' Move row to random position
rstAuthors.Move count, adBookmarkFirst
' Remember the mystery row
target = rstAuthors.Bookmark

count = 0
rstAuthors.MoveFirst
' Loop through recordset
Do Until rstAuthors.EOF
    result = rstAuthors.CompareBookmarks(rstAuthors.Bookmark, tar

If result = adCompareNotEqual Then
    Debug.Print "Row "; count; ": Bookmarks are not equal."
ElseIf result = adCompareNotComparable Then
    Debug.Print "Row "; count; ": Bookmarks are not comparable
Else
    Select Case result
        Case adCompareLessThan
            strAnswer = "less than"
        Case adCompareEqual
            strAnswer = "equal to"
        Case adCompareGreaterThan
            strAnswer = "greater than"
        Case Else
            strAnswer = "in error comparing to"
    End Select
    'show the results row-by-row
    Debug.Print "Row position " & count & " is " & strAnswer &
End If

    count = count + 1
    rstAuthors.MoveNext
Loop

' clean up
rstAuthors.Close
Cnxn.Close
Set rstAuthors = Nothing
Set Cnxn = Nothing

```

```
Exit Sub

ErrorHandler:
    ' clean up
    If Not rstAuthors Is Nothing Then
        If rstAuthors.State = adStateOpen Then rstAuthors.Close
    End If
    Set rstAuthors = Nothing

    If Not Cnxn Is Nothing Then
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
    Set Cnxn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If

End Sub
'EndCompareBookmarksVB
```

See Also

[CompareBookmarks Method](#) | [CompareEnum](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ConnectionString, ConnectionTimeout, and State Properties Example (VB)

This example demonstrates different ways of using the [ConnectionString](#) property to open a [Connection](#) object. It also uses the [ConnectionTimeout](#) property to set a connection timeout period, and the [State](#) property to check the state of the connections. The GetState function is required for this procedure to run.

```
'BeginConnectionStringVB

    'To integrate this code replace
    'the database, DSN or Data Source values

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn1 As ADODB.Connection
    Dim Cnxn2 As ADODB.Connection
    Dim Cnxn3 As ADODB.Connection
    Dim Cnxn4 As ADODB.Connection

    ' Open a connection without using a Data Source Name (DSN)
    Set Cnxn1 = New ADODB.Connection
    Cnxn1.ConnectionString = "Provider='sqloledb';Data Source='MySQL
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn1.Open
    MsgBox "Cnxn1 state: " & GetState(Cnxn1.State)

    ' Open a connection using a DSN and ODBC tags
    ' It is assumed that you have create DSN 'Pubs' with a user nam
    ' 'MyUserId' and password as 'MyPassword'.
    Set Cnxn2 = New ADODB.Connection
    Cnxn2.ConnectionString = "Data Source='Pubs';" & _
        "User ID='MyUserId';Password='MyPassword';"
    Cnxn2.ConnectionTimeout = 30
    Cnxn2.Open
    MsgBox "Cnxn2 state: " & GetState(Cnxn2.State)

    ' Open a connection using a DSN and OLE DB tags
```

```

' It is assumed that you have create DSN 'Pubs1' with windows a
Set Cnxn3 = New ADODB.Connection
Cnxn3.ConnectionString = "Data Source='Pubs1';"
Cnxn3.Open
MsgBox "Cnxn2 state: " & GetState(Cnxn3.State)

' Open a connection using a DSN and individual
' arguments instead of a connection string
' It is assumed that you have create DSN 'Pubs' with a user nam
' 'MyUserId' and password as 'MyPassword'.
Set Cnxn4 = New ADODB.Connection
Cnxn4.Open "Pubs", "MyUserId", "MyPassword"
MsgBox "Cnxn4 state: " & GetState(Cnxn4.State)

' clean up
Cnxn1.Close
Cnxn2.Close
Cnxn3.Close
Cnxn4.Close
Set Cnxn1 = Nothing
Set Cnxn2 = Nothing
Set Cnxn3 = Nothing
Set Cnxn4 = Nothing
Exit Sub

```

ErrorHandler:

```

' clean up
If Not Cnxn1 Is Nothing Then
    If Cnxn1.State = adStateOpen Then Cnxn1.Close
End If
Set Cnxn1 = Nothing

If Not Cnxn2 Is Nothing Then
    If Cnxn2.State = adStateOpen Then Cnxn2.Close
End If
Set Cnxn2 = Nothing

If Not Cnxn3 Is Nothing Then
    If Cnxn3.State = adStateOpen Then Cnxn3.Close
End If
Set Cnxn3 = Nothing

If Not Cnxn4 Is Nothing Then
    If Cnxn4.State = adStateOpen Then Cnxn4.Close
End If
Set Cnxn4 = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

```

```
End Sub

Public Function GetState(intState As Integer) As String

    Select Case intState
        Case adStateClosed
            GetState = "adStateClosed"
        Case adStateOpen
            GetState = "adStateOpen"
    End Select

End Function
'EndConnectionStringVB
```

See Also

[Connection Object](#) | [ConnectionString Property](#) | [ConnectionTimeout Property](#) | [State Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ConvertToString Method Example (VB)

```
'BeginConvertToStringVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' to integrate this code replace the server name
    ' in the CreateObject call

    ' RDS variables
    Dim rdsDS As RDS.DataSpace
    Dim rdsDC As RDS.DataControl
    Dim rdsDF As Object
    ' recordset and connection variables
    Dim rsAuthors As ADODB.Recordset
    Dim strSQLAuthors As String
    Dim strCnxn As String
    Dim varString As Variant

    ' Create a DataSpace object
    Set rdsDS = New RDS.DataSpace
    ' Create a DataFactory object
    Set rdsDF = rdsDS.CreateObject("RDS.Server.DataFactory", "http://

    ' Get all of the Author records

    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    strSQLAuthors = "SELECT * FROM Authors"
    Set rsAuthors = rdsDF.Query(strCnxn, strSQLAuthors)
    ' Show results
    Debug.Print "Old RDS recordset count:" & rsAuthors.RecordCount

    ' Convert the recordset into a MIME formatted string
    varString = rdsDF.ConvertToString(rsAuthors)
    Debug.Print "Recordset as MIME format string:"
    Debug.Print varString
```

```

    ' Convert string value back into an ADO Recordset
Set rdsDC = New RDS.DataControl
rdsDC.SQL = varString
rdsDC.ExecuteOptions = adcExecSync
rdsDC.FetchOptions = adcFetchUpFront
rdsDC.Refresh
    ' Show results
Debug.Print "New ADO recordset count:" & rdsDC.Recordset.RecordC

    ' clean up
rsAuthors.Close
Set rsAuthors = Nothing
Set rdsDC = Nothing
Set rdsDS = Nothing
Set rdsDC = Nothing

```

ErrorHandler:

```

If Not rsAuthors Is Nothing Then
    If rsAuthors.State = adStateOpen Then rsAuthors.Close
End If
Set rsAuthors = Nothing
Set rdsDC = Nothing
Set rdsDS = Nothing
Set rdsDC = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

```

```

End Sub
'EndConvertToStringVB

```

See Also

[ConvertToString Method \(RDS\)](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

CopyRecord, CopyTo, and SaveToFile Methods Example (VB)

This example demonstrates how to create copies of a file using [Stream](#) or [Record](#) objects. One copy is made to a Web folder for Internet publishing. Other properties and methods shown include [Stream Type](#), **Open**, [LoadFromFile](#), and [Record Open](#).

```
'BeginCopyRecordVB
```

```
'Note:
```

```
' This sample requires that "C:\checkmrk.wmf" and  
' "http://MyServer/mywmf.wmf" exist.
```

```
Option Explicit
```

```
Private Sub Form_Load()
```

```
    On Error GoTo ErrorHandler
```

```
    ' Declare variables
```

```
    Dim strPicturePath, strStreamPath, strStream2Path, _  
        strRecordPath, strStreamURL, strRecordURL As String
```

```
    Dim objStream, objStream2 As Stream
```

```
    Dim objRecord As Record
```

```
    Dim objField As Field
```

```
    ' Instantiate objects
```

```
    Set objStream = New Stream
```

```
    Set objStream2 = New Stream
```

```
    Set objRecord = New Record
```

```
    ' Initialize path and URL strings
```

```
    strPicturePath = "C:\checkmrk.wmf"
```

```
    strStreamPath = "C:\mywmf.wmf"
```

```
    strStreamURL = "URL=http://MyServer/mywmf.wmf"
```

```
    strStream2Path = "C:\checkmrk2.wmf"
```

```
    strRecordPath = "C:\mywmf.wmf"
```

```
    strRecordURL = "http://MyServer/mywmf2.wmf"
```

```
    ' Load the file into the stream
```

```
    objStream.Open
```

```
    objStream.Type = adTypeBinary
```

```
    objStream.LoadFromFile (strPicturePath)
```

```

' Save the stream to a new path and filename
objStream.SaveToFile strStreamPath, adSaveCreateOverWrite

' Copy the contents of the first stream to a second stream
objStream2.Open
objStream2.Type = adTypeBinary
objStream.CopyTo objStream2

' Save the second stream to a different path
objStream2.SaveToFile strStream2Path, adSaveCreateOverWrite

' Because strStreamPath is a Web Folder, open a Record on the UR
objRecord.Open "", strStreamURL

' Display the Fields of the record
For Each objField In objRecord.Fields
    Debug.Print objField.Name & ": " & objField.Value
Next

' Copy the record to a new URL
objRecord.CopyRecord "", strRecordURL, , , adCopyOverWrite

' Load each copy of the graphic into Image controls for viewing
Image1.Picture = LoadPicture(strPicturePath)
Image2.Picture = LoadPicture(strStreamPath)
Image3.Picture = LoadPicture(strStream2Path)
Image4.Picture = LoadPicture(strRecordPath)

' clean up
objStream.Close
objStream2.Close
objRecord.Close
Set objStream = Nothing
Set objStream2 = Nothing
Set objRecord = Nothing
Exit Sub

```

ErrorHandler:

```

' clean up
If Not objStream Is Nothing Then
    If objStream.State = adStateOpen Then objStream.Close
End If
Set objStream = Nothing

If Not objStream2 Is Nothing Then
    If objStream2.State = adStateOpen Then objStream2.Close
End If
Set objStream2 = Nothing

```

```
If Not objRecord Is Nothing Then
    If objRecord.State = adStateOpen Then objRecord.Close
End If
Set objRecord = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndCopyRecordVB
```

See Also

[CopyRecord Method](#) | [CopyTo Method](#) | [LoadFromFile Method](#) | [Open Method \(ADO Record\)](#) | [Open Method \(ADO Stream\)](#) | [Record Object](#) | [SaveToFile Method](#) | [Stream Object](#) | [Type Property \(ADO Stream\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Count Property Example (VB)

This example demonstrates the [Count](#) property with two collections in the *Employee* database. The property obtains the number of objects in each collection, and sets the upper limit for loops that enumerate these collections. Another way to enumerate these collections without using the **Count** property would be to use For Each...Next statements.

```
'BeginCountVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' recordset and connection variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLEmployees As String
    Dim strCnxn As String

    Dim intLoop As Integer

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Northwind';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset with data from Employee table
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployees = "Employees"
    'rstEmployees.Open strSQLEmployee, Cnxn, , , adCmdTable
    rstEmployees.Open strSQLEmployees, Cnxn, adOpenForwardOnly, adLo
    'the above two lines opening the recordset are identical as
    'the default values for CursorType and LockType arguments match

    ' Print information about Fields collection
    Debug.Print rstEmployees.Fields.Count & " Fields in Employee"

    For intLoop = 0 To rstEmployees.Fields.Count - 1
        Debug.Print " " & rstEmployees.Fields(intLoop).Name
```

```

Next intLoop

' Print information about Properties collection
Debug.Print rstEmployees.Properties.Count & " Properties in Empl

For intLoop = 0 To rstEmployees.Properties.Count - 1
    Debug.Print "    " & rstEmployees.Properties(intLoop).Name
Next intLoop

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndCountVB

```

See Also

[Count Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

CreateRecordset Method Example (VB)

You can create a [Recordset](#) object and specify the column information. You can then insert data into the **Recordset** object; the underlying [rowset](#) buffers the inserts.

The following code example shows how to define a **Recordset** by using the [RDSServer.DataFactory](#) object. You can also do this with the [RDS.DataControl](#) object.

```
'BeginRsDefineShapeVB
Sub Main()
    On Error GoTo ErrorHandler

    Dim ADF As RDSServer.DataFactory
    Dim vntRecordShape(3)
    Dim vntField1Shape(3)
    Dim vntField2Shape(3)
    Dim vntField3Shape(3)
    Dim vntField4Shape(3)

    Set ADF = New RDSServer.DataFactory

    ' For each field, specify the name,
    ' type, size, and nullability.

    vntField1Shape(0) = "Name"      ' Column name.
    vntField1Shape(1) = CInt(129)   ' Column type.
    vntField1Shape(2) = CInt(40)    ' Column size.
    vntField1Shape(3) = False       ' Nullable?

    vntField2Shape(0) = "Age"
    vntField2Shape(1) = CInt(3)
    vntField2Shape(2) = CInt(-1)
    vntField2Shape(3) = True

    vntField3Shape(0) = "DateOfBirth"
    vntField3Shape(1) = CInt(7)
    vntField3Shape(2) = CInt(-1)
    vntField3Shape(3) = True
```

```

vntField4Shape(0) = "Balance"
vntField4Shape(1) = CInt(6)
vntField4Shape(2) = CInt(-1)
vntField4Shape(3) = True

' Put all fields into an array of arrays.
vntRecordShape(0) = vntField1Shape
vntRecordShape(1) = vntField2Shape
vntRecordShape(2) = vntField3Shape
vntRecordShape(3) = vntField4Shape

' Use the RDSServer.DataFactory to create an empty
' recordset. It takes an array of variants where
' every element is itself another array of
' variants, one for every column required in the
' recordset.
' The elements of the inner array are the column's
' name, type, size, and nullability.
'
' NOTE: You could just use the RDS.DataControl object
' instead of the RDSServer.DataFactory object. In
' that case, the following code would be Set NewRS
' = ADC1.CreateRecordset(vntRecordShape)
Dim NewRs As ADODB.Recordset
Set NewRs = ADF.CreateRecordSet(vntRecordShape)

Dim fields(3)
fields(0) = vntField1Shape(0)
fields(1) = vntField2Shape(0)
fields(2) = vntField3Shape(0)
fields(3) = vntField4Shape(0)

' Populate the new recordset with data values.
Dim fieldVals(3)

' Use AddNew to add the records.
fieldVals(0) = "Joe"
fieldVals(1) = 5
fieldVals(2) = CDate(#1/5/1996#)
fieldVals(3) = 123.456
NewRs.AddNew fields, fieldVals

fieldVals(0) = "Mary"
fieldVals(1) = 6
fieldVals(2) = CDate(#6/5/1996#)
fieldVals(3) = 31
NewRs.AddNew fields, fieldVals

fieldVals(0) = "Alex"
fieldVals(1) = 13

```

```

fieldVals(2) = CDate(#1/6/1996#)
fieldVals(3) = 34.0001
NewRs.AddNew fields, fieldVals

fieldVals(0) = "Susan"
fieldVals(1) = 13
fieldVals(2) = CDate(#8/6/1996#)
fieldVals(3) = 0#
NewRs.AddNew fields, fieldVals

NewRs.MoveFirst

' Set the newly created and populated Recordset to
' the SourceRecordset property of the
' RDS.DataControl to bind to visual controls
Dim ADC1 As RDS.DataControl
Set ADC1 = New RDS.DataControl
Set ADC1.SourceRecordset = NewRs

'Clean up
If NewRs.State = adStateOpen Then NewRs.Close
Set NewRs = Nothing
Set ADC1 = Nothing
Set ADF = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not NewRs Is Nothing Then
    If NewRs.State = adStateOpen Then NewRs.Close
End If
Set NewRs = Nothing
Set ADC1 = Nothing
Set ADF = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndRsDefineShapeVB

```

ADO 2.5 Samples 

CursorType, LockType, and EditMode Properties Example (VB)

This example demonstrates setting the [CursorType](#) and [LockType](#) properties before opening a [Recordset](#). It also shows the value of the [EditMode](#) property under various conditions. The **EditModeOutput** function is required for this procedure to run.

```
'BeginEditModeVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' recordset variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim SQLEmployees As String

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' set recordset properties through object refs
    ' instead of through arguments to Open method
    Set rstEmployees = New ADODB.Recordset
    Set rstEmployees.ActiveConnection = Cnxn
    rstEmployees.CursorLocation = adUseClient
    rstEmployees.CursorType = adOpenStatic
    rstEmployees.LockType = adLockBatchOptimistic

    ' open recordset with data from Employee table
    SQLEmployees = "employee"
    rstEmployees.Open SQLEmployees, , , , adCmdTable

    ' Show the EditMode property under different editing states
    rstEmployees.AddNew
```

```

rstEmployees!emp_id = "T-T55555M"
rstEmployees!fname = "temp_fname"
rstEmployees!lname = "temp_lname"
    'call function below
    'to output results to debug window
EditModeOutput "After AddNew:", rstEmployees.EditMode
rstEmployees.UpdateBatch
EditModeOutput "After UpdateBatch:", rstEmployees.EditMode
rstEmployees!fname = "test"
EditModeOutput "After Edit:", rstEmployees.EditMode
rstEmployees.Close

' Delete new record because this is a demonstration
Cnxn.Execute "DELETE FROM employee WHERE emp_id = 'T-T55555M'"

' clean up
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

```

ErrorHandler:

```

' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

```

End Sub

```

Public Function EditModeOutput(strTemp As String, _
    intEditMode As Integer)

' Print report based on the value of the EditMode
' property
Debug.Print strTemp
Debug.Print "    EditMode = ";

Select Case intEditMode
    Case adEditNone
        Debug.Print "adEditNone"

```

```
Case adEditInProgress
    Debug.Print "adEditInProgress"
Case adEditAdd
    Debug.Print "adEditAdd"
End Select
```

```
End Function
'EndEditModeVB
```

See Also

[CursorType Property](#) | [CursorTypeEnum](#) | [EditMode Property](#) | [EditModeEnum](#) | [LockType Property](#) | [LockTypeEnum](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Delete Method Example (VB)

This example uses the [Delete](#) method to remove a specified record from a [Recordset](#).

```
'BeginDeleteVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim rstRoySched As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLRoySched As String

    Dim strMsg As String
    Dim strTitleID As String
    Dim intLoRange As Integer
    Dim intHiRange As Integer
    Dim intRoyalty As Integer

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' open RoySched table with cursor client-side
    Set rstRoySched = New ADODB.Recordset
    rstRoySched.CursorLocation = adUseClient
    rstRoySched.CursorType = adOpenStatic
    rstRoySched.LockType = adLockBatchOptimistic
    rstRoySched.Open "SELECT * FROM roysched WHERE royalty = 20", st

    ' Prompt for a record to delete
    strMsg = "Before delete there are " & rstRoySched.RecordCount &
        " titles with 20 percent royalty:" & vbCrLf & vbCrLf

    Do While Not rstRoySched.EOF
        strMsg = strMsg & rstRoySched!title_id & vbCrLf
        rstRoySched.MoveNext
```

Loop

```
strMsg = strMsg & vbCr & vbCr & "Enter the ID of a record to del  
strTitleID = UCase(InputBox(strMsg))
```

```
If strTitleID = "" Then  
    Err.Raise 1, , "You didn't enter any value for the record ID  
End If
```

```
' Move to the record and save data so it can be restored  
rstRoySched.Filter = "title_id = '" & strTitleID & "'"
```

```
If rstRoySched.RecordCount < 1 Then  
    Err.Raise 1, , "There is no record for the record ID you ent  
End If
```

```
intLoRange = rstRoySched!lorange  
intHiRange = rstRoySched!hirange  
intRoyalty = rstRoySched!royalty
```

```
' Delete the record  
rstRoySched.Delete  
rstRoySched.UpdateBatch
```

```
' Show the results  
rstRoySched.Filter = adFilterNone  
rstRoySched.Requery  
strMsg = ""  
strMsg = "After delete there are " & rstRoySched.RecordCount & _  
    " titles with 20 percent royalty:" & vbCr & vbCr  
Do While Not rstRoySched.EOF  
    strMsg = strMsg & rstRoySched!title_id & vbCr  
    rstRoySched.MoveNext
```

```
Loop  
MsgBox strMsg
```

```
' Restore the data because this is a demonstration  
rstRoySched.AddNew  
rstRoySched!title_id = strTitleID  
rstRoySched!lorange = intLoRange  
rstRoySched!hirange = intHiRange  
rstRoySched!royalty = intRoyalty  
rstRoySched.UpdateBatch
```

```
' clean up  
rstRoySched.Close  
Set rstRoySched = Nothing  
Exit Sub
```

ErrorHandler:

```
' clean up
If Not rstRoySched Is Nothing Then
    If rstRoySched.State = adStateOpen Then rstRoySched.Close
End If
Set rstRoySched = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndDeleteVB
```

See Also

[Delete Method \(ADO Recordset\)](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

DeleteRecord and MoveRecord Methods Example (VB)

This example demonstrates how to copy, move, edit, and delete the contents of a text file published to a Web folder. Other properties and methods used include [GetChildren](#), [ParentURL](#), [Source](#), and [Flush](#).

```
'BeginDeleteRecordVB

'Note:
' IIS must be running for this sample to work. To
' use this sample you must:
'
' 1. create folders named "test" and "test2"
'    in the root web folder of http://MyServer
'
' 2. Create a text file named "test2.txt" in the
'    "test" folder.
' 3. Replace "MyServer" with the appropriate web
'    server name.

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection and recordset variables
    Dim rsDestFolder As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String

    ' file as record variables
    Dim rcFile As ADODB.Record
    Dim rcDestFile As ADODB.Record
    Dim rcDestFolder As ADODB.Record
    Dim objStream As Stream

    ' file variables
    Dim strFile As String
    Dim strDestFile As String
    Dim strDestFolder As String

    ' instantiate variables
    Set rsDestFolder = New ADODB.Recordset
    Set rcDestFolder = New ADODB.Record
```

```

Set rcFile = New ADODB.Record
Set rcDestFile = New ADODB.Record
Set objStream = New ADODB.Stream

' open a record on a text file
Set Cnxn = New ADODB.Connection
strCnxn = "url=http://MyServer/"
Cnxn.Open strCnxn
strFile = "test/test2.txt"
rcFile.Open strFile, Cnxn, adModeReadWrite, adOpenIfExists Or ad
Debug.Print Cnxn

' edit the contents of the text file
objStream.Open rcFile, , adOpenStreamFromRecord

Debug.Print "Source: " & strCnxn & rcFile.Source
Debug.Print "Original text: " & objStream.ReadText

objStream.Position = 0
objStream.WriteText "Newer Text. "
objStream.Position = 0

Debug.Print "New text: " & objStream.ReadText

' reset the stream object
objStream.Flush
objStream.Close
rcFile.Close

' reopen record to see new contents of text file
rcFile.Open strFile, Cnxn, adModeReadWrite, adOpenIfExists Or ad
objStream.Open rcFile, adModeReadWrite, adOpenStreamFromRecord

Debug.Print "Source: " & strCnxn & rcFile.Source
Debug.Print "Edited text: " & objStream.ReadText

' copy the file to another folder
strDestFile = "test2/test1.txt"
rcFile.CopyRecord "", "http://MyServer/" & strDestFile, "", "",

' delete the original file
rcFile.DeleteRecord

' move the file from the subfolder back to original location
strDestFolder = "test2/"
rcDestFolder.Open strDestFolder, Cnxn ', adOpenIfExists 'Or adC
Set rsDestFolder = rcDestFolder.GetChildren
rsDestFolder.MoveFirst

' position current record at on the correct file

```

```

Do While Not (rsDestFolder.EOF Or rsDestFolder(0) = "test1.txt")
    rsDestFolder.MoveNext
Loop

' open a record on the correct row of the recordset
rcDestFile.Open rsDestFolder, Cnxn

' do the move
rcDestFile.MoveRecord "", "http://MyServer/" & strFile, "", "",

' clean up
rsDestFolder.Close
Cnxn.Close
Set rsDestFolder = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rsDestFolder Is Nothing Then
    If rsDestFolder.State = adStateOpen Then rsDestFolder.Close
End If
Set rsDestFolder = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndDeleteRecordVB

```

See Also

[DeleteRecord Method](#) | [Flush Method](#) | [GetChildren Method](#) | [MoveRecord Method](#) | [ParentURL Property](#) | [Source Property \(ADO Record\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState Properties Example (VB)

This example triggers an error, traps it, and displays the [Description](#), [HelpContext](#), [HelpFile](#), [NativeError](#), [Number](#), [Source](#), and [SQLState](#) properties of the resulting [Error](#) object.

```
'BeginDescriptionVB
Public Sub Main()

    Dim Cnxn As ADODB.Connection
    Dim Err As ADODB.Error
    Dim strError As String

    On Error GoTo ErrorHandler

    ' Intentionally trigger an error
    Set Cnxn = New ADODB.Connection
    Cnxn.Open "nothing"

    Set Cnxn = Nothing
    Exit Sub

ErrorHandler:

    ' Enumerate Errors collection and display
    ' properties of each Error object
    For Each Err In Cnxn.Errors
        strError = "Error #" & Err.Number & vbCr & _
            " " & Err.Description & vbCr & _
            " (Source: " & Err.Source & ")" & vbCr & _
            " (SQL State: " & Err.SQLState & ")" & vbCr & _
            " (NativeError: " & Err.NativeError & ")" & vbCr
        If Err.HelpFile = "" Then
            strError = strError & " No Help file available"
        Else
            strError = strError & _
                " (HelpFile: " & Err.HelpFile & ")" & vbCr & _
                " (HelpContext: " & Err.HelpContext & ")" & _
                vbCr & vbCr
        End If
    End For
End Sub
```

```
        Debug.Print strError
    Next

    Resume Next
End Sub
'EndDescriptionVB
```

See Also

[Description Property](#) | [Error Object](#) | [HelpContext Property](#) | [HelpFile Property](#) | [NativeError Property](#) | [Number Property](#) | [Source Property \(ADO Error\)](#) | [SQLState Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

EOS and LineSeparator Properties and SkipLine Method Example (VB)

This example demonstrates how to manipulate text streams one line at a time. The effect of changing the line separator from the default carriage return/linefeed (**adCRLF**) to simply linefeed (**adLF**) or carriage return (**adCR**) is shown.

```
'BeginSkipLineVB
Private Sub cmdSkipLine_Click()
    On Error GoTo ErrorHandler

    'Declare variables
    Dim i As Integer
    Dim objStream As Stream
    Dim strLine, strChar As String

    'Instantiate and open stream
    Set objStream = New Stream
    objStream.Open

    'Set line separator to line feed
    objStream.LineSeparator = adLF

    'Load text content of list box into stream
    'One line at a time
    For i = 0 To (List1.ListCount - 1)
        objStream.WriteText List1.List(i), adWriteLine
    Next

    'Display the entire stream
    Debug.Print "Whole Stream:"
    objStream.Position = 0
    Debug.Print objStream.ReadText

    'Display the first line
    Debug.Print "First Line:"
    objStream.Position = 0
    strLine = objStream.ReadText(adReadLine)
    Debug.Print strLine
    Debug.Print "Line length: " + Str(Len(strLine))

    'Skip a line, then display another line
    Debug.Print "Third Line:"
```

```

objStream.SkipLine
strLine = objStream.ReadText(adReadLine)
Debug.Print strLine
Debug.Print "Line length: " + Str(Len(strLine))

'Switch line separator to carriage return
'All items from list will be considered one line
'Assuming no CRs have been loaded into stream
Debug.Print "Whole Stream/First Line:"
objStream.Position = 0
objStream.LineSeparator = adCR
strLine = objStream.ReadText(adReadLine)
Debug.Print strLine
Debug.Print "Line length: " + Str(Len(strLine))
Debug.Print "Stream size: " + Str(objStream.Size)

'Use EOS to Determine End of Stream
Debug.Print "Character by character:"
objStream.Position = 0
Do Until objStream.EOS
    strChar = objStream.ReadText(1)
    Debug.Print strChar
Loop

' clean up
objStream.Close
Set objStream = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not objStream Is Nothing Then
    If objStream.State = adStateOpen Then objStream.Close
End If
Set objStream = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub

Private Sub Form_Load()
    List1.AddItem "This is the first line"
    List1.AddItem "This is the second line"
    List1.AddItem "This is the third line"
End Sub
'EndSkipLineVB

```

See Also

[EOS Property](#) | [LineSeparator Property](#) | [SkipLine Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Execute, Requery, and Clear Methods Example (VB)

This example demonstrates the **Execute** method when run from both a [Command](#) object and a [Connection](#) object. It also uses the [Requery](#) method to retrieve current data in a [Recordset](#), and the [Clear](#) method to clear the contents of the [Errors](#) collection. (The **Errors** collection is accessed via the **Connection** object of the [ActiveConnection](#) property of the [Recordset](#).) The ExecuteCommand and PrintOutput procedures are required for this procedure to run.

```
'BeginExecuteVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo Err_Execute

    ' connection, command, and recordset variables
    Dim Cnxn As ADODB.Connection
    Dim cmdChange As ADODB.Command
    Dim rstTitles As ADODB.Recordset
    Dim Err As ADODB.Error
    Dim strSQLChange As String
    Dim strSQLRestore As String
    Dim strSQLTitles
    Dim strCnxn As String

    ' Define two SQL statements to execute as command text
    strSQLChange = "UPDATE Titles SET Type = 'self_help' WHERE Type
    strSQLRestore = "UPDATE Titles SET Type = 'psychology' WHERE Typ

    ' Open connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Create command object
    Set cmdChange = New ADODB.Command
```

```

Set cmdChange.ActiveConnection = Cnxn
cmdChange.CommandText = strSQLChange

' Open titles table
Set rstTitles = New ADODB.Recordset
strSQLTitles = "titles"
rstTitles.Open strSQLTitles, Cnxn, , , adCmdTable

' Print report of original data
Debug.Print _
    "Data in Titles table before executing the query"
PrintOutput rstTitles

' Call the ExecuteCommand subroutine below to execute cmdChange
ExecuteCommand cmdChange, rstTitles

' Print report of new data
Debug.Print _
    "Data in Titles table after executing the query"
PrintOutput rstTitles

' Use the Connection object's execute method to
' execute SQL statement to restore data and trap for
' errors, checking the Errors collection if necessary
Cnxn.Execute strSQLRestore, , adExecuteNoRecords

' Retrieve the current data by requerying the recordset
rstTitles.Requery

' Print report of restored data using sub from below
Debug.Print "Data after executing the query to restore the origi
PrintOutput rstTitles

' clean up
rstTitles.Close
Cnxn.Close
Set rstTitles = Nothing
Set Cnxn = Nothing
Exit Sub

```

Err_Execute:

```

' Notify user of any errors that result from
' executing the query
If rstTitles.ActiveConnection.Errors.Count >= 0 Then
    For Each Err In rstTitles.ActiveConnection.Errors
        MsgBox "Error number: " & Err.Number & vbCr & _
            Err.Description
    Next Err
End If

```

```

' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then rstTitles.Close
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub

Public Sub ExecuteCommand(cmdTemp As ADODB.Command, rstTemp As ADODB

    Dim Err As Error

    ' Run the specified Command object and trap for
    ' errors, checking the Errors collection
    On Error GoTo Err_Execute
    cmdTemp.Execute
    On Error GoTo 0

    ' Retrieve the current data by requerying the recordset
    rstTemp.Requery

Exit Sub

Err_Execute:

    ' Notify user of any errors that result from
    ' executing the query
    If rstTemp.ActiveConnection.Errors.Count > 0 Then
        For Each Err In rstTemp.ActiveConnection.Errors
            MsgBox "Error number: " & Err.Number & vbCr & _
                Err.Description
        Next Err
    End If

    Resume Next

End Sub

Public Sub PrintOutput(rstTemp As ADODB.Recordset)

    ' Enumerate Recordset

```

```
Do While Not rstTemp.EOF
    Debug.Print " " & rstTemp!Title & _
        ", " & rstTemp!Type
    rstTemp.MoveNext
Loop

End Sub
'EndExecuteVB
```

See Also

[Clear Method](#) | [Command Object](#) | [Connection Object](#) | [Error Object](#) | [Execute Method \(ADO Command\)](#) | [Execute Method \(ADO Connection\)](#) | [Recordset Object](#) | [Requery Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Filter and RecordCount Properties Example (VB)

This example opens a **Recordset** on the Publishers table in the *Pubs* database. It then uses the [Filter](#) property to limit the number of visible records to those publishers in a particular country/region. The **RecordCount** property is used to show the difference between the filtered and unfiltered recordsets.

```
'BeginFilterVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' recordset variables
    Dim rstPublishers As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim SQLPublishers As String

    ' criteria variables
    Dim intPublisherCount As Integer
    Dim strCountry As String
    Dim strMessage As String

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' open recordset with data from Publishers table
    Set rstPublishers = New ADODB.Recordset
    SQLPublishers = "publishers"
    rstPublishers.Open SQLPublishers, strCnxn, adOpenStatic, , adCmd

    intPublisherCount = rstPublishers.RecordCount

    ' get user input
    strCountry = Trim(InputBox("Enter a country to filter on (e.g. U
```

```

If strCountry <> "" Then
    ' open a filtered Recordset object
    rstPublishers.Filter = "Country =" & strCountry & ""

    If rstPublishers.RecordCount = 0 Then
        MsgBox "No publishers from that country."
    Else
        ' print number of records for the original recordset
        ' and the filtered recordset
        strMessage = "Orders in original recordset: " & _
            vbCr & intPublisherCount & vbCr & _
            "Orders in filtered recordset (Country = '" & _
            strCountry & "'): " & vbCr & _
            rstPublishers.RecordCount
        MsgBox strMessage
    End If
End If

' clean up
rstPublishers.Close
Cnxn.Close
Set rstPublishers = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstPublishers Is Nothing Then
    If rstPublishers.State = adStateOpen Then rstPublishers.Clos
End If
Set rstPublishers = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

End Sub
'EndFilterVB

```

Note When you know the data you want to select, it's usually more efficient to open a **Recordset** with an SQL statement. This example shows how you can create just one **Recordset** and obtain records from a particular

country/region.

```
'BeginFilter2VB
```

```
    'To integrate this code  
    'replace the data source and initial catalog values  
    'in the connection string
```

```
Public Sub Main()
```

```
    On Error GoTo ErrorHandler
```

```
    Dim rstPublishers As ADODB.Recordset  
    Dim Cnxn As ADODB.Connection  
    Dim strSQLPublishers As String  
    Dim strCnxn As String
```

```
    ' open connection  
    Set Cnxn = New ADODB.Connection  
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _  
        "Initial Catalog='Pubs';Integrated Security='SSPI';"  
    Cnxn.Open strCnxn
```

```
    ' open recordset with criteria from Publishers table  
    Set rstPublishers = New ADODB.Recordset  
    strSQLPublishers = "SELECT * FROM publishers WHERE Country = 'US'  
    rstPublishers.Open strSQLPublishers, Cnxn, adOpenStatic, adLockR
```

```
    ' print recordset  
    rstPublishers.MoveFirst  
    Do While Not rstPublishers.EOF  
        Debug.Print rstPublishers!pub_name & ", " & rstPublishers!co  
        rstPublishers.MoveNext  
    Loop
```

```
    ' clean up  
    rstPublishers.Close  
    Cnxn.Close  
    Set rstPublishers = Nothing  
    Set Cnxn = Nothing  
    Exit Sub
```

```
ErrorHandler:
```

```
    ' clean up  
    If Not rstPublishers Is Nothing Then  
        If rstPublishers.State = adStateOpen Then rstPublishers.Clos  
    End If  
    Set rstPublishers = Nothing  
  
    If Not Cnxn Is Nothing Then
```

```
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
    Set Cnxn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndFilter2VB
```

See Also

[Filter Property](#) | [RecordCount Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Find Method Example (VB)

This example uses the [Recordset](#) object's [Find](#) method to locate and count the number of business titles in the *Pubs* database. The example assumes the underlying [provider](#) does not support similar functionality.

```
'BeginFindVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection and recordset variables
    Dim Cnxn As New ADODB.Connection
    Dim rstTitles As New ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLTitles As String

    ' record variables
    Dim mark As Variant
    Dim count As Integer

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' open recordset with default parameters which are
    ' sufficient to search forward through a Recordset
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "SELECT title_id FROM titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenStatic, adLockReadOnly,

    count = 0
    rstTitles.Find "title_id LIKE 'BU%'"

    Do While Not rstTitles.EOF
        'continue if last find succeeded
        Debug.Print "Title ID: "; rstTitles!title_id
        'count the last title found
        count = count + 1
```

```

        ' note current position
        mark = rstTitles.Bookmark
        rstTitles.Find "title_id LIKE 'BU'", 1, adSearchForward, ma
        ' above code skips current record to avoid finding the same
        ' last arg (bookmark) is redundant because Find searches fro
Loop

Debug.Print "The number of business titles is " & count

' clean up
rstTitles.Close
Cnxn.Close
Set rstTitles = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then rstTitles.Close
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndFindVB

```

See Also

[Find Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

GetRows Method Example (VB)

This example uses the [GetRows](#) method to retrieve a specified number of rows from a [Recordset](#) and to fill an array with the resulting data. The **GetRows** method will return fewer than the desired number of rows in two cases: either if [EOF](#) has been reached, or if **GetRows** tried to retrieve a record that was deleted by another user. The function returns **False** only if the second case occurs. The `GetRowsOK` function is required for this procedure to run.

```
'BeginGetRowsVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection and recordset variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLEmployees As String
    Dim strCnxn As String
    ' array variable
    Dim arrEmployees As Variant
    ' detail variables
    Dim strMessage As String
    Dim intRows As Integer
    Dim intRecord As Integer

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' open recordset client-side to enable RecordCount
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployees = "SELECT fName, lName, hire_date FROM Employee"
    rstEmployees.Open strSQLEmployees, Cnxn, adOpenStatic, adLockRea

    ' get user input for number of rows
    Do
        strMessage = "Enter number of rows to retrieve:"
```

```

    intRows = Val(InputBox(strMessage))

    ' if bad user input exit the loop
    If intRows <= 0 Then
        MsgBox "Please enter a positive number", vbOKOnly, "Not
    ' if number of requested records is over the total
    ElseIf intRows > rstEmployees.RecordCount Then
        MsgBox "Not enough records in Recordset to retrieve " &
        vbOKOnly, "Over the available total"
    Else
        Exit Do
    End If
Loop

    ' else put the data in an array and print
    arrEmployees = rstEmployees.GetRows(intRows)

    Dim x As Integer, y As Integer

    For x = 0 To intRows - 1
        For y = 0 To 2
            Debug.Print arrEmployees(y, x) & " ";
        Next y
        Debug.Print vbCrLf
    Next x

    ' clean up
    rstEmployees.Close
    Cnxn.Close
    Set rstEmployees = Nothing
    Set Cnxn = Nothing
    Exit Sub

ErrorHandler:
    ' clean up
    If Not rstEmployees Is Nothing Then
        If rstEmployees.State = adStateOpen Then rstEmployees.Close
    End If
    Set rstEmployees = Nothing

    If Not Cnxn Is Nothing Then
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
    Set Cnxn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndGetRowsVB

```

See Also

[GetRows Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

GetString Method Example (VB)

This example demonstrates the [GetString](#) method.

Assume you are debugging a data access problem and want a quick, simple way of printing the current contents of a small [Recordset](#).

```
'BeginGetStringVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection variables
    Dim Cnxn As ADODB.Connection
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String
    Dim varOutput As Variant

    ' specific variables
    Dim strPrompt As String
    Dim strState As String

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' get user input
    strPrompt = "Enter a state (CA, IN, KS, MD, MI, OR, TN, UT): "
    strState = Trim(InputBox(strPrompt, "GetString Example"))

    ' open recordset
    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "SELECT au_fname, au_lname, address, city FROM A
        "WHERE state = '" & strState & "'"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnl

    If Not rstAuthors.EOF Then
        ' Use all defaults: get all rows, TAB as column delimiter,
```

```

' CARRIAGE RETURN as row delimiter, EMPTY-string as null delimit
varOutput = rstAuthors.GetString(adClipString)
' print output
Debug.Print "State = '" & strState & "'"
Debug.Print "Name           Address           City" & vbC
Debug.Print varOutput
Else
Debug.Print "No rows found for state = '" & strState & "'" &
End If

' clean up
rstAuthors.Close
Cnxn.Close
Set rstAuthors = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstAuthors Is Nothing Then
If rstAuthors.State = adStateOpen Then rstAuthors.Close
End If
Set rstAuthors = Nothing

If Not Cnxn Is Nothing Then
If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndGetStringVB

```

See Also

[GetString Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

IsolationLevel and Mode Properties Example (VB)

This example uses the [Mode](#) property to open an exclusive connection, and the [IsolationLevel](#) property to open a transaction that is conducted in isolation of other transactions.

```
'BeginIsolationLevelVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn As ADODB.Connection
    Dim rstTitles As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLTitles As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Mode = adModeShareExclusive
    Cnxn.IsolationLevel = adXactIsolated
    Cnxn.Open strCnxn

    ' open Titles table
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenDynamic, adLockPessimis

    Cnxn.BeginTrans

    ' Display connection mode
    If Cnxn.Mode = adModeShareExclusive Then
        MsgBox "Connection mode is exclusive."
    Else
        MsgBox "Connection mode is not exclusive."
    End If
```

```

' Display isolation level
If Cnxn.IsolationLevel = adXactIsolated Then
    MsgBox "Transaction is isolated."
Else
    MsgBox "Transaction is not isolated."
End If

' Change the type of psychology titles
Do Until rstTitles.EOF
    If Trim(rstTitles!Type) = "psychology" Then
        rstTitles!Type = "self_help"
        rstTitles.Update
    End If
    rstTitles.MoveNext
Loop

' Print current data in recordset
rstTitles.Requery
Do While Not rstTitles.EOF
    Debug.Print rstTitles!Title & " - " & rstTitles!Type
    rstTitles.MoveNext
Loop

' clean up
rstTitles.Close
Cnxn.RollbackTrans
Cnxn.Close
Set rstTitles = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then rstTitles.Close
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then
        Cnxn.RollbackTrans
        Cnxn.Close
    End If
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub

```

'EndIsolationLevelVB

See Also

[Connection Object](#) | [IsolationLevel Property](#) | [Mode Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Item Property Example (VB)

This example demonstrates how the [Item](#) property accesses members of a collection. The example opens the *Authors* table of the *Pubs* database with a parameterized command.

The parameter in the command issued against the database is accessed from the [Command](#) object's [Parameters](#) collection by index and name. The fields of the returned [Recordset](#) are then accessed from that object's [Fields](#) collection by index and name.

```
'BeginItemVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn As ADODB.Connection
    Dim rstAuthors As ADODB.Recordset
    Dim cmd As ADODB.Command
    Dim prm As ADODB.Parameter
    Dim fld As ADODB.Field
    Dim strCnxn As String

    Dim ix As Integer
    Dim limit As Long
    Dim Column(0 To 8) As Variant

    Set Cnxn = New ADODB.Connection
    Set rstAuthors = New ADODB.Recordset
    Set cmd = New ADODB.Command

    'Set the array with the Authors table field (column) names
    Column(0) = "au_id"
    Column(1) = "au_lname"
    Column(2) = "au_fname"
    Column(3) = "phone"
    Column(4) = "address"
    Column(5) = "city"
    Column(6) = "state"
    Column(7) = "zip"
```

```

Column(8) = "contract"

cmd.CommandText = "SELECT * FROM Authors WHERE state = ?"
Set prm = cmd.CreateParameter("ItemXparm", adChar, adParamInput,
cmd.Parameters.Append prm
' set connection
strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
"Initial Catalog='Pubs';Integrated Security='SSPI';"
Cnxn.Open strCnxn
cmd.ActiveConnection = Cnxn
' open recordset
rstAuthors.Open cmd, , adOpenStatic, adLockReadOnly
'Connection and CommandType are omitted because
'a Command object is provided

Debug.Print "The Parameters collection accessed by index..."
Set prm = cmd.Parameters.Item(0)
Debug.Print "Parameter name = "; prm.Name; ", value = "; prm.
Debug.Print

Debug.Print "The Parameters collection accessed by name..."
Set prm = cmd.Parameters.Item("ItemXparm")
Debug.Print "Parameter name = "; prm.Name; ", value = "; prm.
Debug.Print

Debug.Print "The Fields collection accessed by index..."

rstAuthors.MoveFirst
limit = rstAuthors.Fields.Count - 1
For ix = 0 To limit
    Set fld = rstAuthors.Fields.Item(ix)
    Debug.Print "Field "; ix; ": Name = "; fld.Name; _
        ", Value = "; fld.Value; ""
Next ix

Debug.Print

Debug.Print "The Fields collection accessed by name..."

rstAuthors.MoveFirst
For ix = 0 To 8
    Set fld = rstAuthors.Fields.Item(Column(ix))
    Debug.Print "Field name = "; fld.Name; ", Value = "; fld.v
Next ix

' clean up
rstAuthors.Close
Cnxn.Close
Set rstAuthors = Nothing
Set Cnxn = Nothing

```

```
Exit Sub

ErrorHandler:
' clean up
If Not rstAuthors Is Nothing Then
    If rstAuthors.State = adStateOpen Then rstAuthors.Close
End If
Set rstAuthors = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

Set cmd = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

End Sub
'EndItemVB
```

See Also

[Command Object](#) | [Fields Collection](#) | [Item Property](#) | [Parameters Collection](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

MarshalOptions Property Example (VB)

This example uses the [MarshalOptions](#) property to specify what rows are sent back to the server—All Rows or only Modified Rows.

```
'BeginMarshalOptionsVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLEmployees As String

    Dim strOldFirst As String
    Dim strOldLast As String
    Dim strMessage As String
    Dim strMarshalAll As String
    Dim strMarshalModified As String

    ' Open connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' open recordset with names from Employees table
    ' and set some properties through object refs
    Set rstEmployees = New ADODB.Recordset
    rstEmployees.CursorType = adOpenKeyset
    rstEmployees.LockType = adLockOptimistic
    rstEmployees.CursorLocation = adUseClient

    strSQLEmployees = "SELECT fname, lname FROM Employee ORDER BY ln

    rstEmployees.Open strSQLEmployees, Cnxn, , , adCmdText
    ' empty properties have been set above
```

```

' Store original data
strOldFirst = rstEmployees!fname
strOldLast = rstEmployees!lname

' Change data in edit buffer
rstEmployees!fname = "Linda"
rstEmployees!lname = "Kobara"

' Show contents of buffer and get user input
strMessage = "Edit in progress:" & vbCrLf & _
    " Original data = " & strOldFirst & " " & _
    strOldLast & vbCrLf & " Data in buffer = " & _
    rstEmployees!fname & " " & rstEmployees!lname & vbCrLf & vbCrLf
    "Use Update to replace the original data with " & _
    "the buffered data in the Recordset?"
strMarshalAll = "Would you like to send all the rows " & _
    "in the recordset back to the server?"
strMarshalModified = "Would you like to send only " & _
    "modified rows back to the server?"

If MsgBox(strMessage, vbYesNo) = vbYes Then
    If MsgBox(strMarshalAll, vbYesNo) = vbYes Then
        rstEmployees.MarshalOptions = adMarshalAll
        rstEmployees.Update
    ElseIf MsgBox(strMarshalModified, vbYesNo) = vbYes Then
        rstEmployees.MarshalOptions = adMarshalModifiedOnly
        rstEmployees.Update
    End If
End If

' sShow the resulting data
MsgBox "Data in recordset = " & rstEmployees!fname & " " & _
    rstEmployees!lname

' restore original data because this is a demonstration
If Not (strOldFirst = rstEmployees!fname And _
    strOldLast = rstEmployees!lname) Then
    rstEmployees!fname = strOldFirst
    rstEmployees!lname = strOldLast
    rstEmployees.Update
End If

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

```

ErrorHandler:

```
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndMarshalOptionsVB
```

See Also

[MarshalOptions Property](#) | [MarshalOptionsEnum](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

MaxRecords Property Example (VB)

This example uses the [MaxRecords](#) property to open a [Recordset](#) containing the 10 most expensive titles in the *Titles* table.

```
'BeginMaxRecordsVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim rstTitles As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLTitles As String

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset containing the 10 most expensive
    ' titles in the Titles table
    Set rstTitles = New ADODB.Recordset
    rstTitles.MaxRecords = 10

    strSQLTitles = "SELECT Title, Price FROM Titles ORDER BY Price D
    rstTitles.Open strSQLTitles, strCnxn, adOpenStatic, adLockReadOn

    ' Display the contents of the recordset
    Debug.Print "Top Ten Titles by Price:"

    Do Until rstTitles.EOF
        Debug.Print " " & rstTitles!Title & " - " & rstTitles!Price
        rstTitles.MoveNext
    Loop

    ' clean up
    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
```

```
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then rstTitles.Close
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndMaxRecordsVB
```

See Also

[MaxRecords Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Move Method Example (VB)

This example uses the [Move](#) method to position the record pointer based on user input.

```
'BeginMoveVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection and recordset variables
    Dim rstAuthors As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLAuthors As String
    ' record variables
    Dim varBookmark As Variant
    Dim strCommand As String
    Dim lngMove As Long

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset from Authors table
    Set rstAuthors = New ADODB.Recordset
    rstAuthors.CursorLocation = adUseClient
    ' Use client cursor to allow use of AbsolutePosition property
    strSQLAuthors = "SELECT au_id, au_fname, au_lname, city, state F
    rstAuthors.Open strSQLAuthors, strCnxn, adOpenStatic, adLockOpti

    rstAuthors.MoveFirst

Do
    ' Display information about current record and
    ' ask how many records to move

    strCommand = InputBox( _
        "Record " & rstAuthors.AbsolutePosition & _
```

```

        " of " & rstAuthors.RecordCount & vbCr & _
        "Author: " & rstAuthors!au_fname & _
        " " & rstAuthors!au_lname & vbCr & _
        "Location: " & rstAuthors!city & _
        ", " & rstAuthors!State & vbCr & vbCr & _
        "Enter number of records to Move " & _
        "(positive or negative).")

    ' this is for exiting the loop
    lngMove = CLng(strCommand)

    lngMove = CLng(Val(strCommand))
    If lngMove = 0 Then
        MsgBox "You either entered a non-number or canceled the
        Exit Do
    End If

    ' Store bookmark in case the Move goes too far
    ' forward or backward
    varBookmark = rstAuthors.Bookmark

    ' Move method requires parameter of data type Long
    rstAuthors.Move lngMove

    ' Trap for BOF or EOF
    If rstAuthors.BOF Then
        MsgBox "Too far backward! Returning to current record."
        rstAuthors.Bookmark = varBookmark
    End If
    If rstAuthors.EOF Then
        MsgBox "Too far forward! Returning to current record."
        rstAuthors.Bookmark = varBookmark
    End If
Loop

    ' clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing
    Exit Sub

ErrorHandler:
    ' clean up
    If Not rstAuthors Is Nothing Then
        If rstAuthors.State = adStateOpen Then rstAuthors.Close
    End If
    Set rstAuthors = Nothing

    If Not Cnxn Is Nothing Then

```

```
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
    Set Cnxn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndMoveVB
```

See Also

[Move Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

MoveFirst, MoveLast, MoveNext, and MovePrevious Methods Example (VB)

This example uses the [MoveFirst](#), [MoveLast](#), [MoveNext](#), and [MovePrevious](#) methods to move the record pointer of a [Recordset](#) based on the supplied command. The MoveAny procedure is required for this procedure to run.

```
'BeginMoveFirstVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection and recordset variables
    Dim rstAuthors As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLAuthors
    ' record variables
    Dim strMessage As String
    Dim intCommand As Integer

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset from Authors table
    Set rstAuthors = New ADODB.Recordset
    rstAuthors.CursorLocation = adUseClient
    ' Use client cursor to enable AbsolutePosition property
    strSQLAuthors = "Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnl

    ' Show current record information and get user's method choice
    Do
        strMessage = "Name: " & rstAuthors!au_fname & " " & _
```

```

        rstAuthors!au_lname & vbCr & "Record " & _
        rstAuthors.AbsolutePosition & " of " & _
        rstAuthors.RecordCount & vbCr & vbCr & _
        "[1 - MoveFirst, 2 - MoveLast, " & vbCr & _
        "3 - MoveNext, 4 - MovePrevious]"
intCommand = Val(Left(InputBox(strMessage), 1))

' for exiting the loop
If intCommand < 1 Or intCommand > 4 Then
    MsgBox "You either entered a non-number or canceled the
    Exit Do
End If

' Use specified method while trapping for BOF and EOF
Select Case intCommand
    Case 1
        rstAuthors.MoveFirst
    Case 2
        rstAuthors.MoveLast
    Case 3
        rstAuthors.MoveNext
        If rstAuthors.EOF Then
            MsgBox "Already at end of recordset!"
            rstAuthors.MoveLast
        End If
    Case 4
        rstAuthors.MovePrevious
        If rstAuthors.BOF Then
            MsgBox "Already at beginning of recordset!"
            rstAuthors.MoveFirst
        End If
End Select
Loop

' clean up
rstAuthors.Close
Cnxn.Close
Set rstAuthors = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstAuthors Is Nothing Then
    If rstAuthors.State = adStateOpen Then rstAuthors.Close
End If
Set rstAuthors = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close

```

```
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub

'EndMoveFirstVB
```

See Also

[MoveFirst, MoveLast, MoveNext, and MovePrevious Methods](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

NextRecordset Method Example (VB)

This example uses the [NextRecordset](#) method to view the data in a recordset that uses a compound command statement made up of three separate **SELECT** statements.

```
'BeginNextRecordsetVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection and recordset variables
    Dim rstCompound As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim SQLCompound As String

    Dim intCount As Integer

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open compound recordset
    Set rstCompound = New ADODB.Recordset
    SQLCompound = "SELECT * FROM Authors; " & _
        "SELECT * FROM stores; " & _
        "SELECT * FROM jobs"
    rstCompound.Open SQLCompound, Cnxn, adOpenStatic, adLockReadOnly

    ' Display results from each SELECT statement
    intCount = 1
    Do Until rstCompound Is Nothing
        Debug.Print "Contents of recordset #" & intCount

        Do Until rstCompound.EOF
            Debug.Print , rstCompound.Fields(0), rstCompound.Fields(
                rstCompound.MoveNext
        Loop
```

```

        Set rstCompound = rstCompound.NextRecordset
        intCount = intCount + 1
Loop

' clean up
Cnxn.Close
Set rstCompound = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstCompound Is Nothing Then
    If rstCompound.State = adStateOpen Then rstCompound.Close
End If
Set rstCompound = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndNextRecordsetVB

```

See Also

[NextRecordset Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

NumericScale and Precision Properties Example (VB)

This example uses the [NumericScale](#) and [Precision](#) properties to display the numeric scale and precision of fields in the *Discounts* table of the *Pubs* database.

```
'BeginNumericScaleVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection and recordset variables
    Dim rstDiscounts As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim fldTemp As ADODB.Field
    Dim strCnxn As String
    Dim strSQLDiscounts As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset
    Set rstDiscounts = New ADODB.Recordset
    strSQLDiscounts = "Discounts"
    rstDiscounts.Open strSQLDiscounts, Cnxn, adOpenStatic, adLockRea

    ' Display numeric scale and precision of
    ' numeric and small integer fields
    For Each fldTemp In rstDiscounts.Fields
        If fldTemp.Type = adNumeric Or fldTemp.Type = adSmallInt Then
            MsgBox "Field: " & fldTemp.Name & vbCr & _
                "Numeric scale: " & _
                fldTemp.NumericScale & vbCr & _
                "Precision: " & fldTemp.Precision
        End If
    End For
End Sub
```

```

Next fldTemp

' clean up
rstDiscounts.Close
Cnxn.Close
Set rstDiscounts = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstDiscounts Is Nothing Then
    If rstDiscounts.State = adStateOpen Then rstDiscounts.Close
End If
Set rstDiscounts = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

End Sub
'EndNumericScaleVB

```

See Also

[Field Object](#) | [NumericScale Property](#) | [Parameter Object](#) | [Precision Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Open and Close Methods Example (VB)

This example uses the **Open** and [Close](#) methods on both [Recordset](#) and [Connection](#) objects that have been opened.

```
'BeginOpenVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub OpenX()
    On Error GoTo ErrorHandler

    Dim Cnxn As ADODB.Connection
    Dim rstEmployees As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLEmployees As String
    Dim varDate As Variant

    ' Open connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Open employee table
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployees = "employee"
    rstEmployees.Open strSQLEmployees, Cnxn, adOpenKeyset, adLockOpt

    ' Assign the first employee record's hire date
    ' to a variable, then change the hire date
    varDate = rstEmployees!hire_date
    Debug.Print "Original data"
    Debug.Print "  Name - Hire Date"
    Debug.Print "  " & rstEmployees!fname & " " & _
        rstEmployees!lname & " - " & rstEmployees!hire_date
    rstEmployees!hire_date = #1/1/1900#
    rstEmployees.Update
    Debug.Print "Changed data"
    Debug.Print "  Name - Hire Date"
    Debug.Print "  " & rstEmployees!fname & " " & _
```

```

        rstEmployees!lname & " - " & rstEmployees!hire_date

' Requery Recordset and reset the hire date
rstEmployees.Requery
rstEmployees!hire_date = varDate
rstEmployees.Update
Debug.Print "Data after reset"
Debug.Print "  Name - Hire Date"
Debug.Print "  " & rstEmployees!fname & " " & _
    rstEmployees!lname & " - " & rstEmployees!hire_date

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndOpenVB

```

See Also

[Close Method](#) | [Connection Object](#) | [Open Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#) | [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

OpenSchema Method Example (VB)

This example uses the [OpenSchema](#) method to display the name and type of each table in the *Pubs* database.

```
'BeginOpenSchemaVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub OpenSchemaX()
    On Error GoTo ErrorHandler

    Dim Cnxn As ADODB.Connection
    Dim rstSchema As ADODB.Recordset
    Dim strCnxn As String

    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    Set rstSchema = Cnxn.OpenSchema(adSchemaTables)

    Do Until rstSchema.EOF
        Debug.Print "Table name: " & _
            rstSchema!TABLE_NAME & vbCr & _
            "Table type: " & rstSchema!TABLE_TYPE & vbCr
        rstSchema.MoveNext
    Loop

    ' clean up
    rstSchema.Close
    Cnxn.Close
    Set rstSchema = Nothing
    Set Cnxn = Nothing
    Exit Sub

ErrorHandler:
    ' clean up
    If Not rstSchema Is Nothing Then
        If rstSchema.State = adStateOpen Then rstSchema.Close
    End If
    Set rstSchema = Nothing
```

```

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndOpenSchemaVB

```

This example specifies a TABLE_TYPE query constraint in the **OpenSchema** method **Criteria** argument. As a result, only schema information for the Views specified in the **Pubs** database are returned. The example then displays the name(s) and type(s) of each table(s).

```

'BeginOpenSchema2VB
Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn2 As ADODB.Connection
    Dim rstSchema As ADODB.Recordset
    Dim strCnxn As String

    Set Cnxn2 = New ADODB.Connection
    strCnxn = "Provider=sqloledb;" & _
        "Data Source=MySqlServer;Initial Catalog=Pubs;Integrated Sec
Cnxn2.Open strCnxn

    Set rstSchema = Cnxn2.OpenSchema(adSchemaTables, Array(Empty, Err

Do Until rstSchema.EOF
    Debug.Print "Table name: " & _
        rstSchema!TABLE_NAME & vbCr & _
        "Table type: " & rstSchema!TABLE_TYPE & vbCr
    rstSchema.MoveNext
Loop

    ' clean up
    rstSchema.Close
    Cnxn2.Close
    Set rstSchema = Nothing
    Set Cnxn2 = Nothing
    Exit Sub

ErrorHandler:
    ' clean up

```

```
If Not rstSchema Is Nothing Then
    If rstSchema.State = adStateOpen Then rstSchema.Close
End If
Set rstSchema = Nothing

If Not Cnxn2 Is Nothing Then
    If Cnxn2.State = adStateOpen Then Cnxn2.Close
End If
Set Cnxn2 = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndOpenSchema2VB
```

See Also

[OpenSchema Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Optimize Property Example (VB)

This example demonstrates the [Field](#) objects [dynamic Optimize](#) property. The *zip* field of the *Authors* table in the *Pubs* database is not indexed. Setting the [Optimize](#) property to **True** on the *zip* field authorizes ADO to build an index that improves the performance of the [Find](#) method.

```
'BeginOptimizeVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    ' recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String

    ' Open connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' open recordset client-side to enable index creation
    Set rstAuthors = New ADODB.Recordset
    rstAuthors.CursorLocation = adUseClient
    strSQLAuthors = "SELECT * FROM Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnl
    ' Create the index
    rstAuthors!zip.Properties("Optimize") = True
    ' Find Akiko Yokomoto
    rstAuthors.Find "zip = '94595'"

    ' show results
    Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname & "
        rstAuthors!address & " " & rstAuthors!city & " " & rstA
    rstAuthors!zip.Properties("Optimize") = False 'Delete the index

    ' clean up
    rstAuthors.Close
    Cnxn.Close
```

```
Set rstAuthors = Nothing
Set Cnxn = Nothing
Exit Sub
```

ErrorHandler:

```
' clean up
If Not rstAuthors Is Nothing Then
    If rstAuthors.State = adStateOpen Then rstAuthors.Close
End If
Set rstAuthors = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndOptimizeVB
```

See Also

[Field Object](#) | [Optimize Property—Dynamic \(ADO\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

OriginalValue and UnderlyingValue Properties Example (VB)

This example demonstrates the [OriginalValue](#) and [UnderlyingValue](#) properties by displaying a message if a record's underlying data has changed during a [Recordset](#) batch update.

```
'BeginOriginalValueVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    Dim Cnxn As ADODB.Connection
    Dim rstTitles As ADODB.Recordset
    Dim fldType As ADODB.Field
    Dim strCnxn As String
    Dim strSQLTitles As String

    ' Open connection.
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySQLServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset for batch update
    ' using object refs to set properties
    Set rstTitles = New ADODB.Recordset
    Set rstTitles.ActiveConnection = Cnxn
    rstTitles.CursorType = adOpenKeyset
    rstTitles.LockType = adLockBatchOptimistic
    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles

    ' Set field object variable for Type field
    Set fldType = rstTitles!Type

    ' Change the type of psychology titles
    Do Until rstTitles.EOF
        If Trim(fldType) = "psychology" Then
            fldType = "self_help"
        End If
    Loop
End Sub
```

```

        End If
        rstTitles.MoveNext
Loop

' Similate a change by another user by updating
' data using a command string
Cnxn.Execute "UPDATE Titles SET type = 'sociology' " & _
    "WHERE type = 'psychology'"

'Check for changes
rstTitles.MoveFirst
Do Until rstTitles.EOF
    If fldType.OriginalValue <> fldType.UnderlyingValue Then
        MsgBox "Data has changed!" & vbCrLf & vbCrLf & _
            " Title ID: " & rstTitles!title_id & vbCrLf & _
            " Current value: " & fldType & vbCrLf & _
            " Original value: " & _
            fldType.OriginalValue & vbCrLf & _
            " Underlying value: " & _
            fldType.UnderlyingValue & vbCrLf
    End If
    rstTitles.MoveNext
Loop

' Cancel the update because this is a demonstration
rstTitles.CancelBatch

' Restore original values
Cnxn.Execute "UPDATE Titles SET type = 'psychology' " & _
    "WHERE type = 'sociology'"

' clean up
rstTitles.Close
Cnxn.Close
Set rstTitles = Nothing
Set Cnxn = Nothing
Exit Sub

```

ErrorHandler:

```

' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then rstTitles.Close
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

```

```
    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndOriginalValueVB
```

See Also

[OriginalValue Property](#) | [Recordset Object](#) | [UnderlyingValue Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Prepared Property Example (VB)

This example demonstrates the [Prepared](#) property by opening two [Command](#) objects—one prepared and one not prepared.

```
'BeginPreparedVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn As ADODB.Connection
    Dim cmd1 As ADODB.Command
    Dim cmd2 As ADODB.Command

    Dim strCnxn As String
    Dim strCmd As String
    Dim sngStart As Single
    Dim sngEnd As Single
    Dim sngNotPrepared As Single
    Dim sngPrepared As Single
    Dim intLoop As Integer

    ' Open a connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Create two command objects for the same
    ' command - one prepared and one not prepared
    strCmd = "SELECT title, type FROM Titles ORDER BY type"

    Set cmd1 = New ADODB.Command
    Set cmd1.ActiveConnection = Cnxn
    cmd1.CommandText = strCmd

    Set cmd2 = New ADODB.Command
    Set cmd2.ActiveConnection = Cnxn
    cmd2.CommandText = strCmd
    cmd2.Prepared = True
```

```

' Set a timer, then execute the unprepared
' command 20 times
sngStart = Timer
For intLoop = 1 To 20
    cmd1.Execute
Next intLoop
sngEnd = Timer
sngNotPrepared = sngEnd - sngStart

' Reset the timer, then execute the prepared
' command 20 times
sngStart = Timer
For intLoop = 1 To 20
    cmd2.Execute
Next intLoop
sngEnd = Timer
sngPrepared = sngEnd - sngStart

' Display performance results
MsgBox "Performance Results:" & vbCrLf & _
    "    Not Prepared: " & Format(sngNotPrepared, _
    "##0.000") & " seconds" & vbCrLf & _
    "    Prepared: " & Format(sngPrepared, _
    "##0.000") & " seconds"

' clean up
Cnxn.Close
Set Cnxn = Nothing
Set cmd1 = Nothing
Set cmd2 = Nothing
Exit Sub

ErrorHandler:
' clean up
Set cmd1 = Nothing
Set cmd2 = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndPreparedVB

```

See Also

[Command Object](#) | [Prepared Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Provider and DefaultDatabase Properties Example (VB)

This example demonstrates the [Provider](#) property by opening three [Connection](#) objects using different providers. It also uses the [DefaultDatabase](#) property to set the default database for the Microsoft ODBC Provider.

```
'BeginProviderVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection strings

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn1 As ADODB.Connection
    Dim Cnxn2 As ADODB.Connection
    Dim Cnxn3 As ADODB.Connection
    Dim strCnxn As String

    ' Open a connection using the Microsoft ODBC provider
    Set Cnxn1 = New ADODB.Connection
    Cnxn1.ConnectionString = "driver={SQL Server};server='MySqlServe
        "user_id='MyUserID';password='MyPassword';"
    Cnxn1.Open strCnxn
    Cnxn1.DefaultDatabase = "Pubs"

    ' Display the provider
    MsgBox "Cnxn1 provider: " & Cnxn1.Provider

    ' Open a connection using the Microsoft Jet provider
    Set Cnxn2 = New ADODB.Connection
    Cnxn2.Provider = "Microsoft.Jet.OLEDB.4.0"
    Cnxn2.Open "C:\Program Files\Microsoft Office\Office\Samples\nor
        "MyUserID", "MyPassword"

    ' Display the provider.
    MsgBox "Cnxn2 provider: " & Cnxn2.Provider

    ' Open a connection using the Microsoft SQL Server provider
    Set Cnxn3 = New ADODB.Connection
    Cnxn3.Provider = "sqloledb"
```

```
Cnxn3.Open "Data Source='MySQLServer';" & _  
    "Initial Catalog='Pubs';Integrated Security='SSPI';"
```

```
' Display the provider  
MsgBox "Cnxn3 provider: " & Cnxn3.Provider
```

```
' clean up  
Cnxn1.Close  
Cnxn2.Close  
Cnxn3.Close  
Set Cnxn1 = Nothing  
Set Cnxn2 = Nothing  
Set Cnxn3 = Nothing  
Exit Sub
```

```
ErrorHandler:
```

```
If Not Cnxn1 Is Nothing Then  
    If Cnxn1.State = adStateOpen Then Cnxn1.Close  
End If  
Set Cnxn1 = Nothing
```

```
If Not Cnxn2 Is Nothing Then  
    If Cnxn2.State = adStateOpen Then Cnxn2.Close  
End If  
Set Cnxn2 = Nothing
```

```
If Not Cnxn3 Is Nothing Then  
    If Cnxn3.State = adStateOpen Then Cnxn3.Close  
End If  
Set Cnxn3 = Nothing
```

```
If Err <> 0 Then  
    MsgBox Err.Source & "-->" & Err.Description, , "Error"  
End If
```

```
End Sub  
'EndProviderVB
```

See Also

[Connection Object](#) | [DefaultDatabase Property](#) | [Provider Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Read, ReadText, Write, and WriteText Methods Example (VB)

This example demonstrates how to read the contents of a text box into both a text [Stream](#) and a binary **Stream**. Other properties and methods shown include [Position](#), [Size](#), [Charset](#), and [SetEOS](#).

```
'BeginReadVB
Private Sub cmdRead_Click()
    On Error GoTo ErrorHandler

    'Declare variables
    Dim objStream As Stream
    Dim varA As Variant
    Dim bytA() As Byte
    Dim i As Integer
    Dim strBytes As String

    'Instantiate and Open Stream
    Set objStream = New Stream
    objStream.Open

    'Write the text content of a textbox to the stream
    If Text1.Text = "" Then
        Err.Raise 1, , "The text field is blank."
    End If
    objStream.WriteText Text1.Text

    'Display the text contents and size of the stream
    objStream.Position = 0
    Debug.Print "Default text:"
    Debug.Print objStream.ReadText
    Debug.Print objStream.Size

    'Switch character set and display
    objStream.Position = 0
    objStream.Charset = "Windows-1252"
    Debug.Print "New Charset text:"
    Debug.Print objStream.ReadText
    Debug.Print objStream.Size

    'Switch to a binary stream and display
    objStream.Position = 0
```

```

objStream.Type = adTypeBinary
Debug.Print "Binary:"
Debug.Print objStream.Read
Debug.Print objStream.Size

'Load an array of bytes with the text box text
ReDim bytA(Len(Text1.Text))
For i = 1 To Len(Text1.Text)
    bytA(i - 1) = CByte(Asc(Mid(Text1.Text, i, 1)))
Next

'Write the buffer to the binary stream and display
objStream.Position = 0
objStream.Write bytA()
objStream.SetEOS
objStream.Position = 0
Debug.Print "Binary after Write:"
Debug.Print objStream.Read
Debug.Print objStream.Size

'Switch back to a text stream and display
Debug.Print "Translated back:"
objStream.Position = 0
objStream.Type = adTypeText
Debug.Print objStream.ReadText
Debug.Print objStream.Size

' clean up
objStream.Close
Set objStream = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not objStream Is Nothing Then
    If objStream.State = adStateOpen Then objStream.Close
End If
Set objStream = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndReadVB

```

See Also

[Charset Property](#) | [Position Property](#) | [Read Method](#) | [ReadText Method](#) | [SetEOS](#)

[Method](#) | [Size Property \(ADO Stream\)](#) | [Stream Object](#) | [Write Method](#) | [WriteText Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Refresh Method Example (VB)

This example demonstrates using the [Refresh](#) method to refresh the [Parameters](#) collection for a stored procedure [Command](#) object.

Option Explicit

```
'BeginRefreshVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection strings

    ' connection and recordset variables
    Dim Cnxn As ADODB.Connection
    Dim cmdByRoyalty As ADODB.Command
    Dim rstByRoyalty As ADODB.Recordset
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String
    ' record variables
    Dim intRoyalty As Integer
    Dim strAuthorID As String
    Dim strRoyalty As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open a command object for a stored procedure
    ' with one parameter
    Set cmdByRoyalty = New ADODB.Command
    Set cmdByRoyalty.ActiveConnection = Cnxn
    cmdByRoyalty.CommandText = "byroyalty"
    cmdByRoyalty.CommandType = adCmdStoredProc
    cmdByRoyalty.Parameters.Refresh

    ' Get parameter value, execute the command
    ' and store the results in a recordset
    strRoyalty = InputBox("Enter royalty:")
    If strRoyalty = "" Then
```

```
    Err.Raise 1, , "You either didn't enter royalty or canceled  
End If
```

```
intRoyalty = Trim(strRoyalty)  
cmdByRoyalty.Parameters(1) = intRoyalty  
Set rstByRoyalty = cmdByRoyalty.Execute()
```

```
' Open the Authors table to get author names for display  
Set rstAuthors = New ADODB.Recordset  
strSQLAuthors = "Authors"  
rstAuthors.Open strSQLAuthors, Cnxn, adOpenForwardOnly, adLockPe
```

```
' Print current data in the recordset  
' and add author names  
Debug.Print "Authors with " & intRoyalty & " percent royalty"
```

```
Do Until rstByRoyalty.EOF  
    strAuthorID = rstByRoyalty!au_id  
    Debug.Print "    " & rstByRoyalty!au_id & ", ";  
    rstAuthors.Filter = "au_id = '" & strAuthorID & "'" & ""  
    Debug.Print rstAuthors!au_fname & " "; rstAuthors!au_lname  
    rstByRoyalty.MoveNext
```

```
Loop
```

```
' clean up  
rstByRoyalty.Close  
rstAuthors.Close  
Cnxn.Close  
Set rstByRoyalty = Nothing  
Set rstAuthors = Nothing  
Set Cnxn = Nothing  
Exit Sub
```

```
ErrorHandler:
```

```
' clean up  
If Not rstByRoyalty Is Nothing Then  
    If rstByRoyalty.State = adStateOpen Then rstByRoyalty.Close  
End If  
Set rstByRoyalty = Nothing
```

```
If Not rstAuthors Is Nothing Then  
    If rstAuthors.State = adStateOpen Then rstAuthors.Close  
End If  
Set rstAuthors = Nothing
```

```
If Not Cnxn Is Nothing Then  
    If Cnxn.State = adStateOpen Then Cnxn.Close  
End If  
Set Cnxn = Nothing
```

```
    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndRefreshVB
```

See Also

[Command Object](#) | [Parameters Collection](#) | [Refresh Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Resync Method Example (VB)

This example demonstrates using the [Resync](#) method to refresh data in a static recordset.

```
'BeginResyncVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection strings

Public Sub Main()
    On Error GoTo ErrorHandler

    'connection and recordset variables
    Dim Cnxn As ADODB.Connection
    Dim rstTitles As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLTitles As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset using object refs to set properties
    ' that allow for updates to the database
    Set rstTitles = New ADODB.Recordset
    Set rstTitles.ActiveConnection = Cnxn
    rstTitles.CursorType = adOpenKeyset
    rstTitles.LockType = adLockOptimistic

    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles

    'rstTitles.Open strSQLTitles, Cnxn, adOpenKeyset, adLockPessimis
    'the above line of code passes the same refs as the object refs

    ' Change the type of the first title in the recordset
    rstTitles!Type = "database"

    ' Display the results of the change
    MsgBox "Before resync: " & vbCrLf & vbCrLf & _
        "Title - " & rstTitles!Title & vbCrLf & _
```

```

        "Type - " & rstTitles!Type

    ' Resync with database and redisplay results
    rstTitles.Resync
    MsgBox "After resync: " & vbCr & vbCr & _
        "Title - " & rstTitles!Title & vbCr & _
        "Type - " & rstTitles!Type

    ' clean up
    rstTitles.CancelBatch
    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
    Set Cnxn = Nothing
    Exit Sub

ErrorHandler:
    ' clean up
    If Not rstTitles Is Nothing Then
        If rstTitles.State = adStateOpen Then
            rstTitles.CancelBatch
            rstTitles.Close
        End If
    End If
    Set rstTitles = Nothing

    If Not Cnxn Is Nothing Then
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
    Set Cnxn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndResyncVB

```

See Also

[Recordset Object](#) | [Resync Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Save and Open Methods Example (VB)

These three examples demonstrate how the [Save](#) and [Open](#) methods can be used together.

Assume you are going on a business trip and want to take along a table from a database. Before you go, you access the data as a [Recordset](#) and save it in a transportable form. When you arrive at your destination, you access the **Recordset** as a local, disconnected **Recordset**. You make changes to the **Recordset**, then save it again. Finally, when you return home, you connect to the database again and update it with the changes you made on the road.

First, access and save the *Authors* table.

```
'BeginSaveVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    'recordset and connection variables
    Dim rstAuthors As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLAuthors As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "SELECT au_id, au_lname, au_fname, city, phone F
rstAuthors.Open strSQLAuthors, Cnxn, adOpenDynamic, adLockOptimi

    'For sake of illustration, save the Recordset to a diskette in X
```

```

rstAuthors.Save "c:\Pubs.xml", adPersistXML

' clean up
rstAuthors.Close
Cnxn.Close
Set rstAuthors = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
'clean up
If Not rstAuthors Is Nothing Then
    If rstAuthors.State = adStateOpen Then rstAuthors.Close
End If
Set rstAuthors = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndSaveVB

```

At this point, you have arrived at your destination. You will access the **Authors** table as a local, disconnected **Recordset**. Don't forget you must have the **MSPersist provider** on the machine that you are using in order to access the saved file, a:\Pubs.xml.

```

'BeginSave2VB
Public Sub Main()
    On Error GoTo ErrorHandler

    Dim rst As ADODB.Recordset
    Set rst = New ADODB.Recordset

    'For sake of illustration, we specify all parameters
    rst.Open "c:\Pubs.xml", "Provider=MSPersist;", adOpenForwardOnly

    'Now you have a local, disconnected Recordset - Edit as you desi
    '(In this example the change makes no difference)
    rst.Find "au_lname = 'Carson'"
    If rst.EOF Then
        Debug.Print "Name not found."
    Exit Sub

```

```
End If
```

```
rst!city = "Chicago"  
rst.Update
```

```
'Save changes in ADTG format this time, purely for sake of illus  
'Note that the previous version is still on the diskette, as a:\  
rst.Save "c:\Pubs.adtg", adPersistADTG
```

```
' clean up  
rst.Close  
Set rst = Nothing  
Exit Sub
```

```
ErrorHandler:
```

```
'clean up  
If Not rst Is Nothing Then  
    If rst.State = adStateOpen Then rst.Close  
End If  
Set rst = Nothing
```

```
If Err <> 0 Then  
    MsgBox Err.Source & "-->" & Err.Description, , "Error"  
End If
```

```
End Sub
```

```
'EndSave2VB
```

Finally, you return home. Now update the database with your changes.

```
'BeginSave3VB
```

```
Public Sub Main()
```

```
    On Error GoTo ErrorHandler
```

```
'To integrate this code  
'replace the data source and initial catalog values  
'in the connection string
```

```
Dim Cnxn As New ADODB.Connection  
Dim rst As ADODB.Recordset  
Dim strCnxn As String
```

```
Set rst = New ADODB.Recordset
```

```
' The lock mode is batch optimistic because we are going to  
' use the UpdateBatch method.
```

```
rst.Open "c:\Pubs.adtg", "Provider=MSPersist;", adOpenForwardOnly
```

```
' Connect to the database, associate the Recordset with the con  
' then update the database table with the changed Recordset  
strCnxn = "Provider=SQLOLEDB;Data Source=MySqlServer;Integrated
```

```

Cnxn.Open strCnxn

rst.ActiveConnection = Cnxn
rst.UpdateBatch

' clean up
rst.Close
Cnxn.Close
Set rst = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
'clean up
If Not rst Is Nothing Then
    If rst.State = adStateOpen Then rst.Close
End If
Set rst = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndSave3VB

```

See Also

[Open Method \(ADO Recordset\)](#) | [Recordset Object](#) | [Recordset Persistence](#) | [Save Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Seek Method and Index Property Example (VB)

This example uses the [Recordset](#) object's [Seek](#) method and [Index](#) property in conjunction with a given **Employee ID**, to locate the employee's name in the **Employees** table of the Nwind.mdb database.

```
'BeginSeekVB
Public Sub Main()
    On Error GoTo ErrorHandler

    ' To integrate this code replace the data source
    ' in the connection string

    'recordset and connection variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLEmployees As String

    Dim strID As String
    Dim strPrompt As String
    strPrompt = "Enter an EmployeeID (e.g., 1 to 9)"

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\Offi
Cnxn.Open strCnxn

    ' open recordset server-side for indexing
    Set rstEmployees = New ADODB.Recordset
    rstEmployees.CursorLocation = adUseServer
    strSQLEmployees = "employees"
    rstEmployees.Open strSQLEmployees, strCnxn, adOpenKeyset, adLock

    ' Does this provider support Seek and Index?
    If rstEmployees.Supports(adIndex) And rstEmployees.Supports(adSe
        rstEmployees.Index = "PrimaryKey"
        ' Display all the employees
        rstEmployees.MoveFirst
        Do While rstEmployees.EOF = False
            Debug.Print rstEmployees!EmployeeId; ": "; rstEmploy
```

```

        rstEmployees!LastName
        rstEmployees.MoveNext
    Loop

' Prompt the user for an EmployeeID between 1 and 9
rstEmployees.MoveFirst
Do
    strID = LCase(Trim(InputBox(strPrompt, "Seek Example")))
    ' Quit if strID is a zero-length string (CANCEL, null,
    If Len(strID) = 0 Then Exit Do
    If Len(strID) = 1 And strID >= "1" And strID <= "9" Then
        rstEmployees.Seek Array(strID), adSeekFirstEQ
        If rstEmployees.EOF Then
            Debug.Print "Employee not found."
        Else
            Debug.Print strID; ": Employee='"; rstEmployees!f
            rstEmployees!LastName; "'"
        End If
    End If
Loop
End If

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndSeekVB

```

See Also

[Index Property](#) | [Recordset Object](#) | [Seek Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Sort Property Example (VB)

This example uses the [Recordset](#) object's [Sort](#) property to reorder the rows of a **Recordset** derived from the *Authors* table of the *Pubs* database. A secondary utility routine prints each row.

```
'BeginSortVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' connection and recordset variables
    Dim Cnxn As New ADODB.Connection
    Dim rstAuthors As New ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String

    Dim strTitle As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' open client-side recordset to enable sort method
    Set rstAuthors = New ADODB.Recordset
    rstAuthors.CursorLocation = adUseClient
    strSQLAuthors = "SELECT * FROM Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnl

    ' sort the recordset last name ascending
    rstAuthors.Sort = "au_lname ASC, au_fname ASC"
    ' show output
    Debug.Print "Last Name Ascending:"
    Debug.Print "First Name  Last Name" & vbCrLf

    rstAuthors.MoveFirst
    Do Until rstAuthors.EOF
        Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname
        rstAuthors.MoveNext
    
```

```

Loop

    ' sort the recordset last name descending
    rstAuthors.Sort = "au_lname DESC, au_fname ASC"
    ' show output
    Debug.Print "Last Name Descending"
    Debug.Print "First Name  Last Name" & vbCrLf

Do Until rstAuthors.EOF
    Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname
    rstAuthors.MoveNext
Loop

    ' clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing
    Exit Sub

ErrorHandler:
    ' clean up
    If Not rstAuthors Is Nothing Then
        If rstAuthors.State = adStateOpen Then rstAuthors.Close
    End If
    Set rstAuthors = Nothing

    If Not Cnxn Is Nothing Then
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
    Set Cnxn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndSortVB

```

This is the secondary utility routine that prints the given title, and the contents of the specified **Recordset**.

Attribute VB_Name = "Sort"

See Also

[Recordset Object](#) | [Sort Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Source Property Example (VB)

This example demonstrates the [Source](#) property by opening three [Recordset](#) objects based on different data sources.

```
'BeginSourceVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn As ADODB.Connection
    Dim rstTitles As ADODB.Recordset
    Dim rstPublishers As ADODB.Recordset
    Dim rstPublishersDirect As ADODB.Recordset
    Dim rstTitlesPublishers As ADODB.Recordset
    Dim cmdSQL As ADODB.Command
    Dim strCnxn As String
    Dim strSQL As String

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open a recordset based on a command object
    Set cmdSQL = New ADODB.Command
    Set cmdSQL.ActiveConnection = Cnxn
    strSQL = "Select title, type, pubdate FROM Titles ORDER BY title"
    cmdSQL.CommandText = strSQL
    Set rstTitles = cmdSQL.Execute()

    ' Open a recordset based on a table
    Set rstPublishers = New ADODB.Recordset
    strSQL = "Publishers"
    rstPublishers.Open strSQL, Cnxn, adOpenStatic, adLockReadOnly, a
    'rstPublishers.Open strSQL, Cnxn, , , adCmdTable
    ' the above two lines of code are identical

    ' Open a recordset based on a table
    Set rstPublishersDirect = New ADODB.Recordset
```

```
rstPublishersDirect.Open strSQL, strCnxn, , , adCmdTableDirect
```

```
' Open a recordset based on an SQL string.  
Set rstTitlesPublishers = New ADODB.Recordset  
strSQL = "SELECT title_ID AS TitleID, title AS Title, " & _  
        "publishers.pub_id AS PubID, pub_name AS PubName " & _  
        "FROM publishers INNER JOIN Titles " & _  
        "ON publishers.pub_id = Titles.pub_id " & _  
        "ORDER BY Title"  
rstTitlesPublishers.Open strSQL, strCnxn, , , adCmdText
```

```
' Use the Source property to display the source of each recordse  
MsgBox "rstTitles source: " & vbCrLf & _  
        rstTitles.Source & vbCrLf & vbCrLf & _  
        "rstPublishers source: " & vbCrLf & _  
        rstPublishers.Source & vbCrLf & vbCrLf & _  
        "rstPublishersDirect source: " & vbCrLf & _  
        rstPublishersDirect.Source & vbCrLf & vbCrLf & _  
        "rstTitlesPublishers source: " & vbCrLf & _  
        rstTitlesPublishers.Source
```

```
' clean up  
rstTitles.Close  
rstPublishers.Close  
rstTitlesPublishers.Close  
Cnxn.Close  
Set rstTitles = Nothing  
Set rstPublishers = Nothing  
Set rstTitlesPublishers = Nothing  
Set Cnxn = Nothing  
Exit Sub
```

ErrorHandler:

```
' clean up  
If Not rstTitles Is Nothing Then  
    If rstTitles.State = adStateOpen Then rstTitles.Close  
End If  
Set rstTitles = Nothing  
  
If Not rstPublishers Is Nothing Then  
    If rstPublishers.State = adStateOpen Then rstPublishers.Clos  
End If  
Set rstPublishers = Nothing  
  
If Not rstTitlesPublishers Is Nothing Then  
    If rstTitlesPublishers.State = adStateOpen Then rstTitlesPub  
End If  
Set rstTitlesPublishers = Nothing  
  
If Not Cnxn Is Nothing Then
```

```
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
    Set Cnxn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
'EndSourceVB
```

See Also

[Recordset Object](#) | [Source Property \(ADO Recordset\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

State Property Example (VB)

This example uses the [State](#) property to display a message while [asynchronous](#) connections are opening and asynchronous commands are executing.

```
'BeginStateVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn1 As ADODB.Connection
    Dim Cnxn2 As ADODB.Connection
    Dim cmdChange As ADODB.Command
    Dim cmdRestore As ADODB.Command
    Dim strCnxn As String
    Dim strSQL As String

    ' Open two asynchronous connections, displaying
    ' a message while connecting
    Set Cnxn1 = New ADODB.Connection
    Set Cnxn2 = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"

    Cnxn1.Open strCnxn, , , adAsyncConnect
    Do Until Cnxn1.State <> adStateConnecting
        Debug.Print "Opening first connection...."
    Loop

    Cnxn2.Open strCnxn, , , adAsyncConnect
    Do Until Cnxn2.State <> adStateConnecting
        Debug.Print "Opening second connection...."
    Loop

    ' Create two command objects
    Set cmdChange = New ADODB.Command
    cmdChange.ActiveConnection = Cnxn1
    strSQL = "UPDATE Titles SET type = 'self_help' WHERE type = 'psy"
    cmdChange.CommandText = strSQL

    Set cmdRestore = New ADODB.Command
```

```
cmdRestore.ActiveConnection = Cnxn2
strSQL = "UPDATE Titles SET type = 'psychology' WHERE type = 'se
cmdRestore.CommandText = strSQL
```

```
' Executing the commands, displaying a message
' while they are executing
cmdChange.Execute , , adAsyncExecute
Do Until cmdChange.State <> adStateExecuting
    Debug.Print "Change command executing...."
Loop
```

```
cmdRestore.Execute , , adAsyncExecute
Do Until cmdRestore.State <> adStateExecuting
    Debug.Print "Restore command executing...."
Loop
```

```
' clean up
Cnxn1.Close
Cnxn2.Close
Set Cnxn1 = Nothing
Set Cnxn2 = Nothing
Exit Sub
```

ErrorHandler:

```
' clean up
If Not Cnxn1 Is Nothing Then
    If Cnxn1.State = adStateOpen Then Cnxn1.Close
End If
Set Cnxn1 = Nothing
```

```
If Not Cnxn2 Is Nothing Then
    If Cnxn2.State = adStateOpen Then Cnxn2.Close
End If
Set Cnxn2 = Nothing
```

```
If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
```

```
End Sub
'EndStateVB
```

See Also

[Recordset Object](#) | [State Property](#)

ADO 2.5 Samples 

Status Property Example (Field) (VB)

The following example opens a document from a read/write folder using the [Internet Publishing Provider](#). The [Status](#) property of a [Field](#) object of the [Record](#) will first be set to **adFieldPendingInsert**, then be updated to **adFieldOk**.

```
'BeginStatusFieldVB

' to integrate this code replace the values in the source string

Sub Main()

    Dim File As ADODB.Record
    Dim strFile As String
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String

    Set Cnxn = New ADODB.Connection
    strCnxn = "url=http://MyServer/"
    Cnxn.Open strCnxn

    Set File = New ADODB.Record
    strFile = "Folder/FileName"
    ' Open a read/write document
    File.Source = strFile
    File.ActiveConnection = Cnxn
    File.Mode = adModeReadWrite
    File.Open

    Debug.Print "Append a couple of fields"

    File.Fields.Append "chektest:fld1", adWChar, 42, adFldUpdatable,
    File.Fields.Append "chektest:fld2", adWChar, 42, adFldUpdatable,

    Debug.Print "status for the fields"
    Debug.Print File.Fields("chektest:fld1").Status 'adfldpendinginse
    Debug.Print File.Fields("chektest:fld2").Status 'adfldpendinginse

    'turn off error-handling to verify field status
    On Error Resume Next

    File.Fields.Update

    Debug.Print "Update succeeds"
    Debug.Print File.Fields("chektest:fld1").Status 'adfldpendinginse
```

```

Debug.Print File.Fields("chektest:fld2").Status 'adfldpendinginse
' resume default error-handling
On Error GoTo 0

' clean up
File.Close
Cnxn.Close
Set File = Nothing
Set Cnxn = Nothing

End Sub
'EndStatusFieldVB

```

The following example deletes a known **Field** from a **Record** opened from a document. The **Status** property will first be set to **adFieldOK**, then **adFieldPendingUnknown**.

```

'BeginStatusField2VB

' to integrate this code replace the values in the source string

Sub Main()

    Dim File As ADODB.Record
    Dim fld As ADODB.Field
    Dim strFile As String

    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String

    ' create connection as a URL
    Set Cnxn = New ADODB.Connection
    strCnxn = "url=http://MyServer/"
    Cnxn.Open strCnxn
    Set File = New ADODB.Record
    strFile = "Folder/FileName"
    ' Open a read/write document
    File.Open strFile, Cnxn, adModeReadWrite
    Debug.Print File.Fields("chektest:fld1").Status ' should be adFld

    ' Delete a field which already exists in the collection
    File.Fields.Delete "chektest:fld1"
    Set fld = File.Fields("chektest:fld1")
    Debug.Print File.Fields("chektest:fld1").Status ' Pending delet

    'turn off error-handling to verify field status
    On Error Resume Next

```

```

File.Fields.Update

Debug.Print "Deleted"
Debug.Print fld.Status    ' Pending unknown

' resume default error-handling
On Error GoTo 0

' clean up
File.Close
Cnxn.Close
Set File = Nothing
Set Cnxn = Nothing

End Sub
'EndStatusField2VB

```

The following code deletes a **Field** from a **Record** opened on a read-only document. **Status** will be set to **adFieldPendingDelete**. At [Update](#), the delete will fail and **Status** will be **adFieldPendingDelete** plus **adFieldPermissionDenied**. [CancelUpdate](#) clears the pending **Status** setting.

```

Sub Main()
Dim File As ADODB.Record
Dim fld As ADODB.Field
Dim Cnxn As ADODB.Connection
Dim strCnxn As String
Dim strFile As String

' create connection as a URL
Set Cnxn = New ADODB.Connection
strCnxn = "url=http://MyServer/"
Cnxn.Open strCnxn

' Open a read/write document
Set File = New ADODB.Record
strFile = "Folder/FileName"
File.Open strFile, Cnxn, adModeReadWrite, adCreateCollection Or a

Debug.Print "Try to delete something without permission"
File.Fields.Delete ("RESOURCE_PARSENAME")
Set fld = File.Fields("RESOURCE_PARSENAME")

Debug.Print "Pending delete"
Debug.Print fld.Status    ' Pending delete
Debug.Print "Delete should fail on Update"

```

```
'turn off error-handling to verify field status
On Error Resume Next

File.Fields.Update    ' Should fail

Debug.Print fld.Status    ' Pending delete plus error
File.Fields.CancelUpdate
Debug.Print fld.Status    ' Okay

' resume default error-handling
On Error GoTo 0

' clean up
File.Close
Cnxn.Close
Set File = Nothing
Set Cnxn = Nothing

End Sub
'EndStatusField3VB
```

See Also

[Field Object](#) | [Record Object](#) | [Status Property \(ADO Field\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Status Property Example (Recordset) (VB)

This example uses the [Status](#) property to display which records have been modified in a batch operation before a batch update has occurred.

```
'BeginStatusRecordsetVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    ' connection and recordset variables
    Dim rstTitles As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLTitles As String

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' open recordset for batch update
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "Titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenKeyset, adLockBatchOpti

    ' change the type of psychology titles
    Do Until rstTitles.EOF
        If Trim(rstTitles!Type) = "psychology" Then rstTitles!Type =
            rstTitles.MoveNext
    Loop

    ' display Title ID and status
    rstTitles.MoveFirst
    Do Until rstTitles.EOF
        If rstTitles.Status = adRecModified Then
            Debug.Print rstTitles!title_id & " - Modified"
        Else
            Debug.Print rstTitles!title_id
        End If
    Loop
End Sub
```

```

        End If
rstTitles.MoveNext
Loop

' clean up
rstTitles.Close
Cnxn.Close
Set rstTitles = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then
        ' Cancel the update because this is a demonstration
        rstTitles.CancelBatch
        rstTitles.Close
    End If
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndStatusRecordsetVB

```

See Also

[Status Property \(ADO Recordset\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

StayInSync Property Example (VB)

This example demonstrates how the [StayInSync](#) property facilitates accessing rows in a hierarchical [Recordset](#).

The outer loop displays each author's first and last name, state, and identification. The appended **Recordset** for each row is retrieved from the [Fields](#) collection and automatically assigned to **rstTitleAuthor** by the **StayInSync** property whenever the parent **Recordset** moves to a new row. The inner loop displays four fields from each row in the appended recordset.

```
'BeginStayInSyncVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code create a DSN called Pubs
    'with a user_id = sa and no password

    Dim Cnxn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim rstTitleAuthor As New ADODB.Recordset
    Dim strCnxn As String

    ' open connection with Data Shape attributes
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=MSDataShape;Data Provider='sqloledb';Data Source=
        'Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' create recordset
    Set rst = New ADODB.Recordset
    rst.StayInSync = True
    rst.Open "SHAPE {select * from Authors} " & _
        "APPEND ( { select * from titleauthor} AS chapTit
        "RELATE au_id TO au_id) ", Cnxn, , , adCmdText

    Set rstTitleAuthor = rst("chapTitleAuthor").Value
    Do Until rst.EOF
        Debug.Print rst!au_fname & " " & rst!au_lname & " " & _
            rst!State & " " & rst!au_id

        Do Until rstTitleAuthor.EOF
            Debug.Print rstTitleAuthor(0) & " " & rstTitleAuthor(1)
                rstTitleAuthor(2) & " " & rstTitleAuthor(3)
```

```

        rstTitleAuthor.MoveNext
    Loop

    rst.MoveNext
Loop

' clean up
rst.Close
Cnxn.Close
Set rst = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rst Is Nothing Then
    If rst.State = adStateOpen Then rst.Close
End If
Set rst = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndStayInSyncVB

```

See Also

[Fields Collection](#) | [Recordset Object](#) | [StayInSync Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Supports Method Example (VB)

This example uses the [Supports](#) method to display the options supported by a recordset opened with different [cursor](#) types. The DisplaySupport procedure is required for this procedure to run.

```
'BeginSupportsVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub Main()
    On Error GoTo ErrorHandler

    ' recordset and connection variables
    Dim rstTitles As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLTitles As String
    ' array variables
    Dim arrCursorType(4) As Integer
    Dim intIndex As Integer

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Fill array with CursorType constants
    arrCursorType(0) = adOpenForwardOnly
    arrCursorType(1) = adOpenKeyset
    arrCursorType(2) = adOpenDynamic
    arrCursorType(3) = adOpenStatic

    ' open recordset using each CursorType and optimistic locking
    For intIndex = 0 To 3
        Set rstTitles = New ADODB.Recordset
        rstTitles.CursorType = arrCursorType(intIndex)
        rstTitles.LockType = adLockOptimistic

        strSQLTitles = "Titles"
        rstTitles.Open strSQLTitles, Cnxn, , , adCmdTable
```

```

    Select Case arrCursorType(intIndex)
        Case adOpenForwardOnly
            Debug.Print "ForwardOnly cursor supports:"
        Case adOpenKeyset
            Debug.Print "Keyset cursor supports:"
        Case adOpenDynamic
            Debug.Print "Dynamic cursor supports:"
        Case adOpenStatic
            Debug.Print "Static cursor supports:"
    End Select

    ' call the DisplaySupport procedure from below
    ' to display the supported options
    DisplaySupport rstTitles

Next intIndex

' clean up
rstTitles.Close
Cnxn.Close
Set rstTitles = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then rstTitles.Close
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndSupportsVB

'BeginSupports2VB
Public Sub DisplaySupport(rstTemp As ADODB.Recordset)

    Dim arrConstants(11) As Long
    Dim blnSupports As Boolean
    Dim intIndex As Integer

```

```

' Fill array with cursor option constants
arrConstants(0) = adAddNew
arrConstants(1) = adApproxPosition
arrConstants(2) = adBookmark
arrConstants(3) = adDelete
arrConstants(4) = adFind
arrConstants(5) = adHoldRecords
arrConstants(6) = adMovePrevious
arrConstants(7) = adNotify
arrConstants(8) = adResync
arrConstants(9) = adUpdate
arrConstants(10) = adUpdateBatch

For intIndex = 0 To 10
    blnSupports = _
        rstTemp.Supports(arrConstants(intIndex))
    If blnSupports Then
        Select Case arrConstants(intIndex)
            Case adAddNew
                Debug.Print "    AddNew"
            Case adApproxPosition
                Debug.Print "    AbsolutePosition and AbsolutePage"
            Case adBookmark
                Debug.Print "    blnkmrk"
            Case adDelete
                Debug.Print "    Delete"
            Case adFind
                Debug.Print "    Find"
            Case adHoldRecords
                Debug.Print "    Holding Records"
            Case adMovePrevious
                Debug.Print "    MovePrevious and Move"
            Case adNotify
                Debug.Print "    Notifications"
            Case adResync
                Debug.Print "    Resyncing data"
            Case adUpdate
                Debug.Print "    Update"
            Case adUpdateBatch
                Debug.Print "    batch updating"
        End Select
    End If
Next intIndex

End Sub
'EndSupports2VB

```

See Also

[Recordset Object](#) | [Supports Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Type Property Example (Field) (VB)

This example demonstrates the [Type](#) property by displaying the name of the constant that corresponds to the value of the [Type](#) property of all the [Field](#) objects in the *Employees* table. The `FieldType` function is required for this procedure to run.

```
'BeginTypeFieldVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    Dim Cnxn As ADODB.Connection
    Dim rstEmployees As ADODB.Recordset
    Dim fld As ADODB.Field
    Dim strCnxn As String
    Dim strSQLEmployee As String
    Dim FieldType As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset with data from Employees table
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployee = "employee"
    rstEmployees.Open strSQLEmployee, Cnxn, , , adCmdTable
    'rstEmployees.Open strSQLEmployee, Cnxn, adOpenStatic, adLockRea
    ' the above two lines of code are identical

    Debug.Print "Fields in Employees Table:" & vbCrLf

    ' Enumerate Fields collection of Employees table
    For Each fld In rstEmployees.Fields
        ' translate field-type code to text
        Select Case fld.Type
            Case adChar
                FieldType = "adChar"
            Case adVarChar
                FieldType = "adVarChar"
```

```

        Case adSmallInt
            FieldType = "adSmallInt"
        Case adUnsignedTinyInt
            FieldType = "adUnsignedTinyInt"
        Case adDBTimeStamp
            FieldType = "adDBTimeStamp"
    End Select
    ' show results
    Debug.Print "  Name: " & fld.Name & vbCr & _
        "  Type: " & FieldType & vbCr
Next fld

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

Set fld = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndTypeFieldVB

```

See Also

[Field Object](#) | [Type Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Type Property Example (Property) (VB)

This example demonstrates the [Type](#) property. It is a model of a utility for listing the names and types of a collection, like [Properties](#), [Fields](#), etc.

We do not need to open the [Recordset](#) to access its **Properties** collection; they come into existence when the **Recordset** object is instantiated. However, setting the [CursorLocation](#) property to **adUseClient** adds several [dynamic properties](#) to the **Recordset** object's **Properties** collection, making the example a little more interesting. For sake of illustration, we explicitly use the [Item](#) property to access each [Property](#) object.

```
'BeginTypePropertyVB
Public Sub Main()
    On Error GoTo ErrorHandler

    ' recordset variables
    Dim rst As ADODB.Recordset
    Dim prop As ADODB.Property
    ' property variables
    Dim ix As Integer
    Dim strMsg As String

    ' create client-side recordset
    Set rst = New ADODB.Recordset
    rst.CursorLocation = adUseClient
    ' enumerate property types
    For ix = 0 To rst.Properties.Count - 1
        Set prop = rst.Properties.Item(ix)
        Select Case prop.Type
            Case adBigInt
                strMsg = "adBigInt"
            Case adBinary
                strMsg = "adBinary"
            Case adBoolean
                strMsg = "adBoolean"
            Case adBSTR
                strMsg = "adBSTR"
            Case adChapter
                strMsg = "adChapter"
            Case adChar
```

```
    strMsg = "adChar"
Case adCurrency
    strMsg = "adCurrency"
Case adDate
    strMsg = "adDate"
Case adDBDate
    strMsg = "adDBDate"
Case adDBTime
    strMsg = "adDBTime"
Case adDBTimeStamp
    strMsg = "adDBTimeStamp"
Case adDecimal
    strMsg = "adDecimal"
Case adDouble
    strMsg = "adDouble"
Case adEmpty
    strMsg = "adEmpty"
Case adError
    strMsg = "adError"
Case adFileTime
    strMsg = "adFileTime"
Case adGUID
    strMsg = "adGUID"
Case adIDispatch
    strMsg = "adIDispatch"
Case adInteger
    strMsg = "adInteger"
Case adIUnknown
    strMsg = "adIUnknown"
Case adLongVarBinary
    strMsg = "adLongVarBinary"
Case adLongVarChar
    strMsg = "adLongVarChar"
Case adLongVarWChar
    strMsg = "adLongVarWChar"
Case adNumeric
    strMsg = "adNumeric"
Case adPropVariant
    strMsg = "adPropVariant"
Case adSingle
    strMsg = "adSingle"
Case adSmallInt
    strMsg = "adSmallInt"
Case adTinyInt
    strMsg = "adTinyInt"
Case adUnsignedBigInt
    strMsg = "adUnsignedBigInt"
Case adUnsignedInt
    strMsg = "adUnsignedInt"
Case adUnsignedSmallInt
```

```

        strMsg = "adUnsignedSmallInt"
    Case adUnsignedTinyInt
        strMsg = "adUnsignedTinyInt"
    Case adUserDefined
        strMsg = "adUserDefined"
    Case adVarBinary
        strMsg = "adVarBinary"
    Case adVarChar
        strMsg = "adVarChar"
    Case adVariant
        strMsg = "adVariant"
    Case adVarNumeric
        strMsg = "adVarNumeric"
    Case adVarWChar
        strMsg = "adVarWChar"
    Case adWChar
        strMsg = "adWChar"
    Case Else
        strMsg = "*UNKNOWN*"
    End Select
    'show results
    Debug.Print "Property " & ix & ": " & prop.Name & _
        ", Type = " & strMsg
Next ix

' clean up
Set rst = Nothing
Exit Sub

ErrorHandler:
' clean up
Set rst = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

End Sub
'EndTypePropertyVB

```

See Also

[Property Object](#) | [Type Property](#)

ADO 2.5 Samples 

Update and CancelUpdate Methods Example (VB)

This example demonstrates the [Update](#) method in conjunction with the [CancelUpdate](#) method.

```
'BeginUpdateVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    ' recordset and connection variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLEmployees As String
    ' buffer variables
    Dim strOldFirst As String
    Dim strOldLast As String
    Dim strMessage As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset to enable changes
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployees = "SELECT fname, lname FROM Employee ORDER BY ln
rstEmployees.Open strSQLEmployees, Cnxn, adOpenKeyset, adLockOpt

    ' Store original data
    strOldFirst = rstEmployees!fname
    strOldLast = rstEmployees!lname
    ' Change data in edit buffer
    rstEmployees!fname = "Linda"
    rstEmployees!lname = "Kobara"

    ' Show contents of buffer and get user input
    strMessage = "Edit in progress:" & vbCrLf & _
```

```

    " Original data = " & strOldFirst & " " & _
    strOldLast & vbCr & " Data in buffer = " & _
    rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr
    "Use Update to replace the original data with " & _
    "the buffered data in the Recordset?"

If MsgBox(strMessage, vbYesNo) = vbYes Then
    rstEmployees.Update
Else
    rstEmployees.CancelUpdate
End If

' show the resulting data
MsgBox "Data in recordset = " & rstEmployees!fname & " " & _
    rstEmployees!lname

' restore original data because this is a demonstration
If Not (strOldFirst = rstEmployees!fname And _
    strOldLast = rstEmployees!lname) Then
    rstEmployees!fname = strOldFirst
    rstEmployees!lname = strOldLast
    rstEmployees.Update
End If

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
' EndUpdateVB

```

This example demonstrates the **Update** method in conjunction with the [AddNew](#) method.

```
' BeginUpdate2VB
Public Sub Main()
    On Error GoTo ErrorHandler

    Dim cnn1 As ADODB.Connection
    Dim rstEmployees As ADODB.Recordset
    Dim strEmpID As String
    Dim strOldFirst As String
    Dim strOldLast As String
    Dim strMessage As String
    Dim strCnn As String

    ' Open a connection.
    Set cnn1 = New ADODB.Connection
        strCnn = "Provider=sqloledb;" & _
            "Data Source=MySqlServer;Initial Catalog=Pubs;Integrated Secu
cnn1.Open strCnn

    ' Open recordset with data from Employees table.
    Set rstEmployees = New ADODB.Recordset
    rstEmployees.CursorType = adOpenKeyset
    rstEmployees.LockType = adLockOptimistic
    rstEmployees.Open "employee", cnn1, , , adCmdTable

    rstEmployees.AddNew
    strEmpID = "B-S55555M"
    rstEmployees!emp_id = strEmpID
    rstEmployees!fname = "Bill"
    rstEmployees!lname = "Sornsinsin"

    ' Show contents of buffer and get user input.
    strMessage = "AddNew in progress:" & vbCrLf & _
        "Data in buffer =" & rstEmployees!emp_id & ", " & _
        rstEmployees!fname & " " & rstEmployees!lname & vbCrLf & vbCrLf
        "Use Update to save buffer to recordset?"

    If MsgBox(strMessage, vbYesNoCancel) = vbYes Then
        rstEmployees.Update
        ' Go to the new record and show the resulting data.
        MsgBox "Data in recordset = " & rstEmployees!emp_id & ", " & _
            rstEmployees!fname & " " & rstEmployees!lname
    Else
        rstEmployees.CancelUpdate
        MsgBox "No new record added."
    End If
```

```
' Delete new data because this is a demonstration.
cnn1.Execute "DELETE FROM employee WHERE emp_id = '" & strEmpID

' clean up
rstEmployees.Close
cnn1.Close
Set rstEmployees = Nothing
Set cnn1 = Nothing
Exit Sub
```

ErrorHandler:

```
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not cnn1 Is Nothing Then
    If cnn1.State = adStateOpen Then cnn1.Close
End If
Set cnn1 = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndUpdate2VB
```

See Also

[CancelUpdate Method](#) | [Recordset Object](#) | [Update Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

UpdateBatch and CancelBatch Methods Example (VB)

This example demonstrates the [UpdateBatch](#) method in conjunction with the [CancelBatch](#) method.

```
'BeginUpdateBatchVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    'connection and recordset variables
    Dim rstTitles As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLTitles As String
    'record variables
    Dim strTitle As String
    Dim strMessage As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' open recordset for batch update
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenKeyset, adLockBatchOpti

    rstTitles.MoveFirst
    ' Loop through recordset and ask user if she wants
    ' to change the type for a specified title.
    Do Until rstTitles.EOF
        If Trim(rstTitles!Type) = "psychology" Then
            strTitle = rstTitles!Title
            strMessage = "Title: " & strTitle & vbCrLf & _
                "Change type to self help?"

            If MsgBox(strMessage, vbYesNo) = vbYes Then
```

```

        rstTitles!Type = "self_help"
    End If
End If

    rstTitles.MoveNext
Loop

' Ask the user if she wants to commit to all the
' changes made above.
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

' clean up
rstTitles.Close
Cnxn.Close
Set rstTitles = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstTitles Is Nothing Then
    If rstTitles.State = adStateOpen Then rstTitles.Close
End If
Set rstTitles = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If

```

```
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndUpdateBatchVB
```

See Also

[CancelBatch Method](#) | [UpdateBatch Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Value Property Example (VB)

This example demonstrates the [Value](#) property with [Field](#) and [Property](#) objects by displaying field and property values for the *Employees* table.

```
'BeginValueVB
Public Sub Main()
    On Error GoTo ErrorHandler

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    ' connection and recordset variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLEmployees As String
    ' field property variables
    Dim fld As ADODB.Field
    Dim prp As ADODB.Property

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    ' Open recordset with data from Employees table
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployees = "employee"
    rstEmployees.Open strSQLEmployees, Cnxn, , , adCmdTable
    'rstEmployees.Open strSQLEmployees, Cnxn, adOpenStatic, adLockRe
    ' the above two lines of code are identical

    Debug.Print "Field values in rstEmployees"

    ' Enumerate the Fields collection of the Employees table
    For Each fld In rstEmployees.Fields
        ' Because Value is the default property of a
        ' Field object, the use of the actual keyword
        ' here is optional.
        Debug.Print "    " & fld.Name & " = " & fld.Value
    Next fld
```

```

Debug.Print "Property values in rstEmployees"

' Enumerate the Properties collection of the Recordset object
For Each prp In rstEmployees.Properties
    Debug.Print "    " & prp.Name & " = " & prp.Value
    ' because Value is the default property of a Property object
    ' use of the actual Value keyword is optional
Next prp

' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
Set Cnxn = Nothing
Exit Sub

ErrorHandler:
' clean up
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not Cnxn Is Nothing Then
    If Cnxn.State = adStateOpen Then Cnxn.Close
End If
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndValueVB

```

See Also

[Field Object](#) | [Property Object](#) | [Value Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Version Property Example (VB)

This example uses the [Version](#) property of a [Connection](#) object to display the current ADO version. It also uses several [dynamic properties](#) to show:

- the current DBMS name and version.
- OLE DB version.
- [provider](#) name and version.
- [ODBC](#) version.
- ODBC driver name and version.

```
'BeginVersionVB
Public Sub Main()
    On Error GoTo ErrorHandler

    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strVersionInfo As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider='sqloledb';Data Source='MySqlServer';" & _
        "Initial Catalog='Pubs';Integrated Security='SSPI';"
    Cnxn.Open strCnxn

    strVersionInfo = "ADO Version: " & Cnxn.Version & vbCr
    strVersionInfo = strVersionInfo & "DBMS Name: " & Cnxn.Properties("DBMS Name") & vbCr
    strVersionInfo = strVersionInfo & "DBMS Version: " & Cnxn.Properties("DBMS Version") & vbCr
    strVersionInfo = strVersionInfo & "OLE DB Version: " & Cnxn.Properties("OLE DB Version") & vbCr
    strVersionInfo = strVersionInfo & "Provider Name: " & Cnxn.Properties("Provider Name") & vbCr
    strVersionInfo = strVersionInfo & "Provider Version: " & Cnxn.Properties("Provider Version") & vbCr

    MsgBox strVersionInfo

    ' clean up
    Cnxn.Close
    Set Cnxn = Nothing
    Exit Sub

ErrorHandler:
    ' clean up
    If Not Cnxn Is Nothing Then
        If Cnxn.State = adStateOpen Then Cnxn.Close
    End If
```

```
Set Cnxn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndVersionVB
```

See Also

[Connection Object](#) | [Version Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ADO Code Examples in Microsoft Visual Basic Scripting Edition

Use the following code examples to learn about how to use the ADO methods when writing in Visual Basic Scripting Edition (VBScript).

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

Methods

- [AddNew Method Example](#)
- [Clone Method Example](#)
- [Delete Method Example](#)
- [Execute, Requery, and Clear Methods Example](#)
- [Move Method Example](#)
- [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods Example](#)
- [Open and Close Methods Example](#)

See Also

[ADO Code Examples in Microsoft Visual Basic](#) | [ADO Code Examples in Microsoft Visual C++](#) | [ADO Code Examples in Microsoft Visual J++](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AddNew Method Example (VBScript)

This example uses the [AddNew](#) method to create a new record with the specified name.

Use the following example in an Active Server Page (ASP). Use **Find** to locate the file Adovbs.inc and place it in the directory you plan to use. Cut and paste the following code to Notepad or another text editor, and save it as **AddNewVBS.asp**. You can view the result in any [client](#) browser.

To exercise the example, add a new record in the HTML form. Click **Add New**. See the [Delete Method Example](#) to remove unwanted records.

```
<!-- BeginAddNewVBS -->
<%@Language = VBScript %>
<% ' use this meta tag instead of adovbs.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E
<HTML>
<HEAD>
  <TITLE>ADO AddNew Method (VBScript)</TITLE>
  <STYLE>
  <!--
  body {
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
  }
  TH {
    background-color: #008080;
    font-family: 'Arial Narrow', 'Arial', sans-serif;
    font-size: xx-small;
    color: white;
  }
  TD {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Arial Narrow', 'Arial', sans-serif;
    font-size: xx-small;
  }
  -->
  </STYLE>
</HEAD>
<BODY>
```

<H1>ADO AddNew Method (VBScript)</H1>

```
<% ' to integrate/test this code replace the
' Data Source value in the Connection string%>
<%
' connection and recordset variables
Dim Cnxn, strCnxn
Dim rsCustomers, strSQLCustomers
Dim fld, Err

' open connection
Set Cnxn = Server.CreateObject("ADODB.Connection")
strCnxn = "Provider='sqloledb';Data Source=" & _
         Request.ServerVariables("SERVER_NAME") & ";" & _
         "Integrated Security='SSPI';Initial Catalog='Northwind';
Cnxn.Open strCnxn

' create and open Recordset using object refs
Set rsCustomers = Server.CreateObject("ADODB.Recordset")
strSQLCustomers = "Customers"

rsCustomers.ActiveConnection = Cnxn
rsCustomers.CursorLocation = adUseClient
rsCustomers.CursorType = adOpenKeyset
rsCustomers.LockType = adLockOptimistic
rsCustomers.Source = strSQLCustomers
rsCustomers.Open

'If this is first time page is open, Form collection
'will be empty when data is entered. run AddNew method
If Not IsEmpty(Request.Form) Then
    If Not Request.Form("CompanyName") = "" Then
        rsCustomers.AddNew
            rsCustomers("CustomerID") = Request.Form("CompanyID")
            rsCustomers("CompanyName") = Request.Form("CompanyNa
            rsCustomers("ContactName") = Request.Form("FirstName
            " " & Request.Form("LastName")
            rsCustomers("Phone") = Request.Form("PhoneNumber")
            rsCustomers("City") = Request.Form("City")
            rsCustomers("Region") = Request.Form("State")
        rsCustomers.Update
        ' check for errors
        If Cnxn.Errors.Count > 0 Then
            For Each Err In Cnxn.Errors
                Response.Write("Error " & Err.SQLState & ": " &
                    Err.Description & " | " & Err.NativeError)
            Next
        Cnxn.Errors.Clear
        rsCustomers.CancelUpdate
```

```

        End If
        'On Error GoTo 0
        rsCustomers.MoveFirst
    End If
End If
%>

<TABLE COLSPAN="8" CELLPADDING=5 BORDER=1 ALIGN="center">
<!-- BEGIN column header row for Customer Table-->
    <TR>
        <TH>Customer ID</TH>
        <TH>Company Name</TH>
        <TH>Contact Name</TH>
        <TH>Phone Number</TH>
        <TH>City</TH>
        <TH>State/Province</TH>
    </TR>

    <% ' show the data
        Do Until rsCustomers.EOF
            Response.Write("<TR>")
            Response.Write("<TD>" & rsCustomers("CustomerID") & "</T
            Response.Write("<TD>" & rsCustomers("CompanyName")& "</T
            Response.Write("<TD>" & rsCustomers("ContactName") & "</
            Response.Write("<TD>" & rsCustomers("Phone") & "</TD>")
            Response.Write("<TD>" & rsCustomers("City") & "</TD>")
            Response.Write("<TD>" & rsCustomers("Region") & "</TD>")
            Response.Write("</TR>")
            rsCustomers.MoveNext
        Loop
    %>
</TABLE>

<HR>

<!--
    Form to enter new record posts variables
    back to this page
-->
<FORM Method=post Action="AddNewVbs.asp" Name=Form>
    <TABLE>
        <TR>
            <TD>Company ID:</TD>
            <TD><INPUT Size="5" Name="CompanyID" maxLength=5 ></TD>
        </TR>
        <TR>
            <TD>Company Name:</TD>
            <TD><INPUT Size="50" Name="CompanyName" ></TD>
        </TR>
    </TABLE>

```

```

<TR>
  <TD>Contact First Name:</TD>
  <TD><INPUT Size="50" Name="FirstName" ></TD>
</TR>
<TR>
  <TD>Contact Last Name:</TD>
  <TD><INPUT Size="50" Name="LastName" ></TD>
</TR>
<TR>
  <TD>Contact Phone:</TD>
  <TD><INPUT Size="50" Name="PhoneNumber" ></TD>
</TR>
<TR>
  <TD>City:</TD>
  <TD><INPUT Size="50" Name="City" ></TD>
</TR>
<TR>
  <TD>State / Province:</TD>
  <TD><INPUT Size="5" Name="State" ></TD>
</TR>
<TR>
  <TD Align="right"><INPUT Type="submit" Value="Add New"><
  <TD Align="left"><INPUT Type="reset" Value="Reset Form">
</TR>
</TABLE>
</FORM>

<%
  ' show connection
  Response.Write("Following is the connection string: <br><br>")
  Response.Write(Cnxn)

  ' Clean up.
  If rsCustomers.State = adStateOpen then
    rsCustomers.Close
  End If
  If Cnxn.State = adStateOpen then
    Cnxn.Close
  End If
  Set rsCustomers=Nothing
  Set Cnxn=Nothing
  Set fld=Nothing
%>

<SCRIPT Language = "VBScript">
Sub Form_OnSubmit
  MsgBox "Sending New Record to Server",, "ADO-ASP _Example"
End Sub
</SCRIPT>
</BODY>

```

```
</HTML>  
<!-- EndAddNewVBS -->
```

See Also

[AddNew Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Clone Method Example (VBScript)

This example uses the [Clone](#) method to create copies of a [Recordset](#) and then lets the user position the record pointer of each copy independently.

Use the following example in an Active Server Page (ASP). This example uses the Northwind database distributed with Microsoft Access. Cut and paste the following code to Notepad or another text editor and save it as **CloneVBS.asp**. You can view the result in any [client](#) browser.

To exercise the example, change the line `RsCustomerList.Source = "Customers"` to `RsCustomerList.Source = "Products"` to count a larger table.

```
<!-- BeginCloneVBS -->
<% Language = VBScript %>
<% ' use this meta tag instead of adovbs.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E
<HTML>
<HEAD>
<TITLE>ADO Clone Method</TITLE>
</HEAD>

<BODY>

<H1 align="center">ADO Clone Method</H1>
<HR>
<% ' to integrate/test this code replace the
' Data Source value in the Connection string%>
<%
' connection and recordset variables
Dim Cnxn, strCnxn
Dim rsCustomers, strSQLCustomers
Dim rsFirst, rsLast, rsCount
Dim rsClone
Dim CloneFirst, CloneLast, CloneCount

' open connection
Set Cnxn = Server.CreateObject("ADODB.Connection")
strCnxn = "Provider='sqloledb';Data Source=" & _
Request.ServerVariables("SERVER_NAME") & ";" & _
"Integrated Security='SSPI';Initial Catalog='Northwind';
Cnxn.Open strCnxn
```

```

' create and open Recordset using object refs
Set rsCustomers = Server.CreateObject("ADODB.Recordset")
strSQLCustomers = "Customers"

rsCustomers.ActiveConnection = Cnxn
rsCustomers.CursorLocation = adUseClient
rsCustomers.CursorType = adOpenKeyset
rsCustomers.LockType = adLockOptimistic
rsCustomers.Source = strSQLCustomers
rsCustomers.Open

rsCustomers.MoveFirst
rsCount = rsCustomers.RecordCount
rsFirst = rsCustomers("CompanyName")
rsCustomers.MoveLast
rsLast = rsCustomers("CompanyName")

' create clone
Set rsClone = rsCustomers.Clone
rsClone.MoveFirst
CloneCount = rsClone.RecordCount
CloneFirst = rsClone("CompanyName")
rsClone.MoveLast
CloneLast = rsClone("CompanyName")
%>

<!-- Display Results -->
<H3>There Are <%=rsCount%> Records in the original recordset</H3>
<H3>The first record is <%=rsFirst%> and the last record is <%=rsLas
<BR><HR>
<H3>There Are <%=CloneCount%> Records in the original recordset</H3>
<H3>The first record is <%=CloneFirst%> and the last record is <%=Cl
<BR><HR>
<H4>Location of OLEDB Database</H4>

<%
' Show location of DSN data source
Response.Write(Cnxn)

' Clean up
If rsCustomers.State = adStateOpen then
    rsCustomers.Close
End If
If rsClone.State = adStateOpen then
    rsClone.Close
End If
If Cnxn.State = adStateOpen then
    Cnxn.Close
End If
Set rsCustomers = Nothing

```

```
        Set rsClone = Nothing
        Set Cnxn = Nothing
%>
</BODY>
</HTML>
<!-- EndCloneVBS -->
```

See Also

[Clone Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Delete Method Example (VBScript)

This example uses the [Delete](#) method to remove a specified record from a [Recordset](#).

Use the following example in an Active Server Page (ASP).

Use **Find** to locate the file Adovbs.inc and place it in the directory you plan to use. Cut and paste the following code into Notepad or another text editor, and save it as **DeleteVBS.asp**. You can view the result in any [client](#) browser.

To exercise the example, try using the [AddNew](#) example first to add some records. Then you can try to delete them. View the result in any client browser.

```
<!-- BeginDeleteVBS -->
<%@ Language=VBScript %>
<% ' use this meta tag instead of ADOVBS.inc%>
<!-- METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2
<HTML>
<HEAD>
<TITLE>ADO Delete Method</TITLE>
</HEAD>
<STYLE>
<!--
TH {
    background-color: #008080;
    font-family: 'Arial Narrow','Arial',sans-serif;
    font-size: xx-small;
    color: white;
}
TD {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Arial Narrow','Arial',sans-serif;
    font-size: xx-small;
}
-->
</STYLE>

<BODY>
<H3>ADO Delete Method</H3>

<%
```

```

' to integrate this code replace the DataSource value in the con
' connection and recordset variables
Dim Cnxn, strCnxn
Dim rsCustomers, strSQLCustomers

' create and open connection
Set Cnxn = Server.CreateObject("ADODB.Connection")
strCnxn="Provider='sqloledb';Data Source=" & _
        Request.ServerVariables("SERVER_NAME") & ";" & _
        "Integrated Security='SSPI';Initial Catalog='Northwind';
Cnxn.Open strCnxn
' create and open recordset
Set rsCustomers = Server.CreateObject("ADODB.Recordset")
strSQLCustomers = "Customers"
rsCustomers.Open strSQLCustomers, Cnxn, adOpenKeyset, adLockOptim

' Move to designated record and delete it
If Not IsEmpty(Request.Form("WhichRecord")) Then
'Get value to move from Form Post method
    Moves = Request.Form("WhichRecord")

    rsCustomers.Move CInt(Moves)
    If Not rsCustomers.EOF or rsCustomers.BOF Then
        ' handle any db errors
        On Error Resume Next
        rsCustomers.Delete 1
        If Cnxn.Errors.Count <> 0 Then
            Response.Write "Cannot delete since there are related re
            Response.End
        End If
        rsCustomers.MoveFirst
        On Error GoTo 0
    Else
        Response.Write "Not a Valid Record Number"
        rsCustomers.MoveFirst
    End If
End If
%>

<!-- BEGIN column header row for Customer Table-->
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<TR>
    <TH>Rec. #</TH>
    <TH>Company Name</TH>
    <TH>Contact Name</TH>
    <TH>City</TH>
</TR>

<%

```

```

' Display ADO Data from Customer Table
' Loop through Recordset adding one row to HTML Table each pass
Dim iCount
iCount = 0
Do Until rsCustomers.EOF %>
<TR>
  <TD> <%= CStr(iCount) %>
  <TD> <%= rsCustomers("CompanyName")%> </TD>
  <TD> <%= rsCustomers("ContactName")%> </TD>
  <TD> <%= rsCustomers("City")%> </TD>
</TR>
<%
  iCount = iCount + 1
  rsCustomers.MoveNext
Loop
%>
</TABLE>

<!-- Do Client side Input Data Validation Move to named record and D
<Center>
<H4>Clicking Button Will Remove Designated Record</H4>
<H5>There are <%=rsCustomers.RecordCount%> Records in this Set</H5>

<Form Method=Post Action="Deletevbs.asp" Name=Form>
  <Input Type=Text Name="WhichRecord" Size=3>
  <Input Type=Button Name=cmdDelete Value="Delete Record">
</Form>

</BODY>

<Script Language = "VBScript">
Sub cmdDelete_OnClick
  If IsNumeric(Document.Form.WhichRecord.Value) Then
    Document.Form.WhichRecord.Value = CInt(Document.Form.WhichReco
    Dim Response
    Response = MsgBox("Are You Sure About Deleting This Record?",

    If Response = vbYes Then
      Document.Form.Submit
    End If
  Else
    MsgBox "You Must Enter a Valid Record Number",,"ADO-ASP Examp1
  End If
End Sub
</Script>

<%
  ' clean up
  If rsCustomers.State = adStateOpen then

```

```
        rsCustomers.Close
    End If
    If Cnxn.State = adStateOpen then
        Cnxn.Close
    End If
%>
</HTML>
<!-- EndDeleteVBS -->
```

See Also

[Delete Method \(ADO Recordset\)](#) | [Recordset Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Execute, Requery, and Clear Methods Example (VBScript)

This example demonstrates the **Execute** method when run from both a [Command](#) object and a [Connection](#) object. It also uses the [Requery](#) method to retrieve current data in a [recordset](#), and the [Clear](#) method to clear the contents of the [Errors](#) collection. The ExecuteCommand and PrintOutput procedures are required for this procedure to run.

Use the following example in an Active Server Page (ASP). Use **Find** to locate the file Adovbs.inc and place it in the directory you plan to use. Cut and paste the following code into Notepad or another text editor, and save it as **ExecuteVBS.asp**. You can view the result in any [client](#) browser.

```
<!-- BeginExecuteVBS -->
<%@ Language=VBScript %>
<% ' use this meta tag instead of ADOVBS.inc%>
<!-- METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2
<HTML>
<HEAD>
<META name="VI60_DefaultClientScript" content=VBScript>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<title>ADO Execute Method</title>
<STYLE>
<!--
BODY {
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
}
.thead {
    background-color: #008080;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
}
.thead2 {
    background-color: #800000;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
```

```

    }
    .tbody {
        text-align: center;
        background-color: #f7efde;
        font-family: 'Verdana','Arial','Helvetica',sans-serif;
        font-size: x-small;
    }
-->
</STYLE>
</HEAD>

<BODY>
<H3>ADO Execute Method</H3>
<HR>
<H4>Recordset Retrieved Using Connection Object</H4>
<!-- Recordsets retrieved using Execute method of Connection and Co
<%
    ' connection, command and recordset variables
    Dim Cnxn, strCnxn
    Dim rsCustomers, strSQLCustomers
    Dim Cmd
    Dim rsProducts, strSQLProducts

    ' create and open connection
    Set Cnxn = Server.CreateObject("ADODB.Connection")
    strCnxn="Provider='sqloledb';Data Source=" & _
        Request.ServerVariables("SERVER_NAME") & ";" & _
        "Integrated Security='SSPI';Initial Catalog='Northwind';
    Cnxn.Open strCnxn
    ' create and open recordset
    Set rsCustomers = Server.CreateObject("ADODB.Recordset")
    strSQLCustomers = "Customers"
    rsCustomers.Open strSQLCustomers, Cnxn, adOpenKeyset, adLockOpti

    '1st Recordset using Connection - Execute
    Set rsCustomers = Cnxn.Execute(strSQLCustomers)

    Set Cmd = Server.CreateObject("ADODB.Command")
    Cmd.ActiveConnection = Cnxn
    strSQLProducts = "SELECT * From Products"
    Cmd.CommandText = strSQLProducts

    '2nd Recordset Cmd - execute
    Set rsProducts = Cmd.Execute
%>
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0 ALIGN=CENTER>
<!-- BEGIN column header row for Customer Table-->
<TR CLASS=thead>
    <TH>Company Name</TH>
    <TH>Contact Name</TH>

```

```

        <TH>City</TH>
</TR>

<!-- Display ADO Data from Customer Table-->
<%
Do While Not rsCustomers.EOF %>
    <TR CLASS=tbody>
        <TD>
            <%= rsCustomers("CompanyName")%>
        </TD>
        <TD>
            <%= rsCustomers("ContactName") %>
        </TD>
        <TD>
            <%= rsCustomers("City")%>
        </TD>
    </TR>
    <%
rsCustomers.MoveNext
Loop
%>
</TABLE>

<HR>
<H4>Recordset Retrieved Using Command Object</H4>

<TABLE CELLPADDING=5 BORDER=0 ALIGN=CENTER WIDTH="80%">

<!-- BEGIN column header row for Product List Table-->
<TR CLASS=thead2>
    <TH>Product Name</TH>
    <TH>Unit Price</TH>
</TR>

<!-- Display ADO Data Product List-->
<% Do Until rsProducts.EOF %>
    <TR CLASS=tbody>
        <TD>
            <%= rsProducts("ProductName")%>
        </TD>
        <TD>
            <%= rsProducts("UnitPrice")%>
        </TD>
    <%
rsProducts.MoveNext
Loop

' clean up
If rsCustomers.State = adStateOpen then

```

```
        rsCustomers.Close
    End If
    If rsProducts.State = adStateOpen then
        rsProducts.Close
    End If
    If Cnxn.State = adStateOpen then
        Cnxn.Close
    End If
    Set Cmd = Nothing
    Set rsCustomers = Nothing
    Set rsProducts = Nothing
    Set Cnxn = Nothing
%>
</TABLE>

</BODY>
</HTML>
<!-- EndExecuteVBS -->
```

See Also

[Clear Method](#) | [Command Object](#) | [Connection Object](#) | [Error Object](#) | [Errors Collection](#) | [Execute Method \(ADO Command\)](#) | [Execute Method \(ADO Connection\)](#) | [Recordset Object](#) | [Requery Method](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Move Method Example (VBScript)

This example uses the [Move](#) method to position the record pointer, based on user input.

Use the following example in an Active Server Page (ASP).

Use **Find** to locate the file Adovbs.inc and place it in the directory you plan to use. Cut and paste the following code to Notepad or another text editor, and save it as **MoveVBS.asp**. You can view the result in any browser.

Try entering a letter or noninteger to see the error handling work.

```
<!-- BeginMoveVBS -->
<%@ Language=VBScript %>
<% ' use this meta tag instead of adovbs.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E
<HTML>
<HEAD>
<TITLE>ADO Move Methods</TITLE>
<STYLE>
<!--
BODY {
    font-family: "MS SANS SERIF", sans-serif;
}
.thead1 {
    background-color: #008080;
    font-family: 'Arial Narrow', 'Arial', sans-serif;
    font-size: x-small;
    color: white;
}
.tbody {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Arial Narrow', 'Arial', sans-serif;
    font-size: x-small;
}
-->
</STYLE>
</HEAD>
<BODY>
<H3>ADO Move Methods</H3>
<% ' to integrate/test this code replace the
```

```

' Data Source value in the Connection string%>
<%
' connection and recordset variables
Dim Cnxn, strCnxn
Dim rsCustomers, strSQLCustomers

' open connection
Set Cnxn = Server.CreateObject("ADODB.Connection")
strCnxn = "Provider='sqloledb';Data Source=" & _
Request.ServerVariables("SERVER_NAME") & ";" & _
"Integrated Security='SSPI';Initial Catalog='Northwind';

Cnxn.Open strCnxn

' create and open Recordset using object refs
Set rsCustomers = Server.CreateObject("ADODB.Recordset")
strSQLCustomers = "Customers"

rsCustomers.ActiveConnection = Cnxn
rsCustomers.CursorLocation = adUseClient
rsCustomers.CursorType = adOpenKeyset
rsCustomers.LockType = adLockOptimistic
rsCustomers.Source = strSQLCustomers
rsCustomers.Open

'Check number of user moves this session and increment by entry
Session("Clicks") = Session("Clicks") + Request.Form("MoveAmount")
Clicks = Session("Clicks")
' Move to last known recordset position plus amount passed
rsCustomers.Move CInt(Clicks)

'Error Handling
If rsCustomers.EOF Then
    Session("Clicks") = rsCustomers.RecordCount
    Response.Write "This is the Last Record"
    rsCustomers.MoveLast
ElseIf rsCustomers.BOF Then
    Session("Clicks") = 1
    rsCustomers.MoveFirst
    Response.Write "This is the First Record"
End If
%>

<H3>Current Record Number is <BR>
<%
If Session("Clicks") = 0 Then Session("Clicks") = 1
Response.Write(Session("Clicks") )%> of <%=rsCustomers.RecordCou
<HR>

```

```

<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>

<!-- BEGIN column header row for Customer Table-->

<TR CLASS=thead1>
  <TD>Company Name</TD>
  <TD>Contact Name</TD>
  <TD>City</TD>
</TR>
  <% 'display%>
  <TR CLASS=tbody>
    <TD> <%= rsCustomers("CompanyName")%> </TD>
    <TD> <%= rsCustomers("ContactName")%></TD>
    <TD> <%= rsCustomers("City")%> </TD>
  </TR>
</TABLE>

<HR>
<Input Type=Button Name=cmdDown Value="&lt; ">
<Input Type=Button Name=cmdUp Value=" &gt;">
<H5>Click Direction Arrows for Previous or Next Record
<BR> <BR>

<FORM Method = Post Action="MoveVbs.asp" Name=Form>
<TABLE>
  <TR>
    <TD><Input Type="Button" Name=Move Value="Move Amount "><
    <TD></TD>
    <TD><Input Type="Text" Size="4" Name="MoveAmount" Value=0
  <TR>
</TABLE>
Click Move Amount to use Move Method<br>
Enter Number of Records to Move + or - </H5>    </FORM>
</BODY>

<Script Language = "VBScript">

Sub Move_OnClick
  ' Make sure move value entered is an integer
  If IsNumeric(Document.Form.MoveAmount.Value)Then
    Document.Form.MoveAmount.Value = CInt(Document.Form.MoveAmount
    Document.Form.Submit
  Else
    MsgBox "You Must Enter a Number", , "ADO-ASP Example"
    Document.Form.MoveAmount.Value = 0
  End If
End Sub

Sub cmdDown_OnClick

```

```
        Document.Form.MoveAmount.Value = -1
        Document.Form.Submit
    End Sub

    Sub cmdUp_OnClick
        Document.Form.MoveAmount.Value = 1
        Document.Form.Submit
    End Sub
</Script>

<%
    ' clean up
    If rsCustomers.State = adStateOpen then
        rsCustomers.Close
    End If
    If Cnxn.State = adStateOpen then
        Cnxn.Close
    End If
%>
</HTML>
<!-- EndMoveVBS -->
```

See Also

[Move Method](#) | [Recordset Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

MoveFirst, MoveLast, MoveNext, and MovePrevious Methods Example (VBScript)

This example uses the [MoveFirst](#), [MoveLast](#), [MoveNext](#), and [MovePrevious](#) methods to move the record pointer of a [Recordset](#) based on the supplied command.

Cut and paste the following code into Notepad or another text editor, and save it as **MoveFirstVBS.asp**. You can view the result in any browser.

```
<!-- BeginMoveFirstVBS -->
<%@ Language=VBScript %>
<% ' use this meta tag instead of adovbs.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E
<HTML>
<HEAD>
<TITLE>ADO MoveNext, MovePrevious, MoveLast, MoveFirst Methods</TITL

<SCRIPT LANGUAGE="VBScript">
<!--
  Sub cmdDown_OnClick
    'Set Values in Form Input Boxes and Submit Form
    Document.form.MoveAction.Value = "MovePrev"
    Document.Form.Submit
  End Sub

  Sub cmdUp_OnClick
    Document.form.MoveAction.Value = "MoveNext"
    Document.Form.Submit
  End Sub

  Sub cmdFirst_OnClick
    Document.form.MoveAction.Value = "MoveFirst"
    Document.Form.Submit
  End Sub

  Sub cmdLast_OnClick
    Document.form.MoveAction.Value = "MoveLast"
    Document.Form.Submit
  End Sub
```

```

//-->
</SCRIPT>
</HEAD>

<body bgcolor="white">
<h1 align="center">ADO MoveNext, MovePrevious <br> MoveLast & MoveFi
<% ' to integrate/test this code replace the
' Data Source value in the Connection string%>
<%
' connection and recordset variables
Dim Cnxn, strCnxn
Dim rsEmployees, strSQLEmployees

' open connection
Set Cnxn = Server.CreateObject("ADODB.Connection")
strCnxn = "Provider='sqloledb';Data Source=" & _
Request.ServerVariables("SERVER_NAME") & ";" & _
"Integrated Security='SSPI';Initial Catalog='Northwind';
Cnxn.Open strCnxn

' create and open Recordset using object refs
Set rsEmployees = Server.CreateObject("ADODB.Recordset")
strSQLEmployees = "Employees"

rsEmployees.ActiveConnection = Cnxn
rsEmployees.CursorLocation = adUseClient
rsEmployees.CursorType = adOpenKeyset
rsEmployees.LockType = adLockOptimistic
rsEmployees.Source = strSQLEmployees
rsEmployees.Open

rsEmployees.MoveFirst

If Not IsEmpty(Request.Form("MoveAction")) Then
strAction = Request.Form("MoveAction")
varPosition = Request.Form("Position")

rsEmployees.AbsolutePosition = varPosition

Select Case strAction

Case "MoveNext"

rsEmployees.MoveNext
If rsEmployees.EOF Then
rsEmployees.MoveLast
strMessage = "Can't move beyond the last record."
End If

Case "MovePrev"

```

```

        rsEmployees.MovePrevious
    If rsEmployees.BOF Then
        rsEmployees.MoveFirst
        strMessage = "Can't move beyond the first record."
    End If

    Case "MoveLast"

        rsEmployees.MoveLast

    Case "MoveFirst"

        rsEmployees.MoveFirst

    End Select
End If

%>

<!-- Display Current Record Number and Recordset Size -->
<h2>Record Number <%=rsEmployees.AbsolutePosition%> of <%=rsEmployee
<hr>
<table cellpadding=5 border=0>
<!-- BEGIN column header row for Customer Table-->
<tr>
    <th>Name</th>
    <th>Hire Date</th>
</tr>

<!--Display ADO Data from Customer Table-->
<tr>
    <td><%= rsEmployees("LastName") & ", " %>
        <%= rsEmployees("FirstName") & " " %></td>
    <td><%= rsEmployees("HireDate")%></td>
</tr>
<tr>
    <td colspan=2><%=strMessage%></td>
</tr>
</table>

<hr>

<form Name="Form" Method="Post" Action="MoveFirstVbs.asp">
<Input Type=Button Name=cmdDown Value="<">
<Input Type=Button Name=cmdUp Value=">">
<BR>
<H3>Click Direction Arrows to Use MovePrevious or MoveNext</H3>
<Input Type=Button Name=cmdFirst Value="First Record">

```

```
<Input Type=Button Name=cmdLast Value="Last Record">

<!-- Use Hidden Form Fields to record values to send to Server -->

<input Type="Hidden" Size="4" Name="MoveAction" Value="Move">
<input Type="Hidden" Size="4" Name="Position" Value="<%= rsEmployees
</form>

<HR>
</BODY>

<%
    ' clean up
    If rsEmployees.State = adStateOpen then
        rsEmployees.Close
    End If
    If Cnxn.State = adStateOpen then
        Cnxn.Close
    End If
%>
</HTML>
<!-- EndMoveFirstVBS -->
```

See Also

[MoveFirst, MoveLast, MoveNext, and MovePrevious Methods](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Open and Close Methods Example (VBScript)

This example uses the [Open](#) and [Close](#) methods on both [Recordset](#) and [Connection](#) objects that have been opened.

Use the following example in an Active Server Page (ASP). Use **Find** to locate the file `Adovbs.inc` and place it in the directory you plan to use. Cut and paste the following code into Notepad or another text editor, and save it as **OpenVBS.asp**. You can view the result in any browser.

```
<!-- BeginOpenVBS -->
<%@ Language=VBScript %>
<% ' use this meta tag instead of adovbs.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E
<HTML>
<HEAD>
<META name="VI60_DefaultClientScript" content=VBScript>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<title>ADO Open Method</title>
<STYLE>
<!--
BODY {
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
}
.thead {
    background-color: #008080;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
}
.thead2 {
    background-color: #800000;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
}
.tbody {
    text-align: center;
    background-color: #f7efde;
```

```

    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
}
-->
</STYLE>
</HEAD>

<BODY>
<H3>ADO Open Method</H3>

<TABLE WIDTH=600 BORDER=0>
<TR>
<TD VALIGN=TOP COLSPAN=3>
<FONT SIZE=2>
<% ' to integrate/test this code replace the
' Data Source value in the Connection string%>
<%
' connection and recordset variables
Dim Cnxn, strCnxn
Dim rsCustomers, strSQLCustomers
Dim rsProducts, strSQLProducts

' open connection
Set Cnxn = Server.CreateObject("ADODB.Connection")
strCnxn = "Provider='sqloledb';Data Source=" & _
Request.ServerVariables("SERVER_NAME") & ";" & _
"Integrated Security='SSPI';Initial Catalog='Northwind';

Cnxn.Open strCnxn

' create and open first Recordset using Connection - execute
Set rsCustomers = Server.CreateObject("ADODB.Recordset")
strSQLCustomers = "SELECT CompanyName, ContactName, City FROM Cu
Set rsCustomers = Cnxn.Execute(strSQLCustomers)

' create and open second Recordset using recordset - open
Set rsProducts = Server.CreateObject("ADODB.Recordset")
strSQLProducts = "SELECT ProductName, UnitPrice FROM Products"
rsProducts.Open strSQLProducts, Cnxn, adOpenDynamic, adLockPessi
%>

<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Customer Table-->
<TR CLASS=thead>
<TD>Company Name</TD>
<TD>Contact Name</TD>
<TD>City</TD>
</TR>

<!--Display ADO Data from Customer Table-->

```

```

<% Do Until rsCustomers.EOF %>
<TR CLASS=tbody>
  <TD> <%=rsCustomers("CompanyName")%> </TD>
  <TD> <%=rsCustomers("ContactName")%></TD>
  <TD> <%=rsCustomers("City")%> </TD>
</TR>
<%rsCustomers.MoveNext
Loop
%>
</TABLE>

<HR>

<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Product List Table-->

<TR CLASS=thead2>
  <TD>Product Name</TD>
  <TD>Unit Price</TD>
</TR>
<!-- Display ADO Data Product List-->
<% Do Until rsProducts.EOF %>
  <TR CLASS=tbody>
    <TD> <%=rsProducts("ProductName")%> </TD>
    <TD> <%=rsProducts("UnitPrice")%> </TD>
  </TR>
  <!-- Next Row = Record -->
<%rsProducts.MoveNext
Loop

' clean up
If rsProducts.State = adStateOpen then
  rsProducts.Close
End If
If rsCustomers.State = adStateOpen then
  rsCustomers.Close
End If
If Cnxn.State = adStateOpen then
  Cnxn.Close
End If
Set rsProducts = Nothing
Set rsCustomers = Nothing
Set Cnxn = Nothing

%>
</TABLE>

</BODY>
</HTML>

```

<!-- EndOpenVBS -->

See Also

[Close Method](#) | [Connection Object](#) | [Open Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ADO Code Examples in Microsoft Visual C++

Use the following code examples to learn how to use the ADO methods, properties, and events when writing in Microsoft Visual C++.

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

Methods

- [AddNew Method Example](#)
- [Append and CreateParameter Methods Example](#)
- [AppendChunk and GetChunk Methods Example](#)
- [BeginTrans, CommitTrans, and RollbackTrans Methods Example](#)
- [Cancel Method Example](#)
- [Clone Method Example](#)
- [CompareBookmarks Method Example](#)
- [Delete Method Example](#)
- [Execute, Requery, and Clear Methods Example](#)
- [Find Method Example](#)
- [GetRows Method Example](#)
- [GetString Method Example](#)
- [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods Example](#)
- [NextRecordset Method Example](#)
- [Open and Close Methods Example](#)
- [OpenSchema Method Example](#)
- [Refresh Method Example](#)
- [Resync Method Example](#)
- [Save and Open Methods Example](#)
- [Seek Method and Index Property Example](#)
- [Supports Method Example](#)
- [Update and CancelUpdate Methods Example](#)
- [UpdateBatch and CancelBatch Methods Example](#)

Properties

- [AbsolutePage, PageCount, and PageSize Properties Example](#)
- [AbsolutePosition and CursorLocation Properties Example](#)
- [ActiveCommand Property Example](#)
- [ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example](#)
- [ActualSize and DefinedSize Properties Example](#)
- [Attributes and Name Properties Example](#)
- [BOF, EOF, and Bookmark Properties Example](#)
- [CacheSize Property Example](#)
- [ConnectionString, ConnectionTimeout, and State Properties Example](#)
- [Count Property Example](#)
- [CursorType, LockType, and EditMode Properties Example](#)
- [Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState Properties Example](#)
- [Filter and RecordCount Properties Example](#)
- [Index Property and Seek Method Example](#)
- [IsolationLevel and Mode Properties Example](#)
- [Item Property Example](#)
- [MarshalOptions Property Example](#)
- [MaxRecords Property Example](#)
- [NumericScale and Precision Properties Example](#)
- [Optimize Property Example](#)
- [OriginalValue and UnderlyingValue Properties Example](#)
- [Prepared Property Example](#)
- [Provider and DefaultDatabase Properties Example](#)
- [Sort Property Example](#)
- [Source Property Example](#)
- [State Property Example](#)
- [Status Property Example](#)
- [StayInSync Property Example](#)
- [Type Property Example \(Fields\)](#)
- [Type Property Example \(Property\)](#)
- [Value Property Example](#)
- [Version Property Example](#)

Other

- [ADO Events Model Example](#)

See Also

[ADO Code Examples in Microsoft Visual Basic](#) | [ADO Code Examples in Microsoft Visual Basic Scripting Edition](#) | [ADO Code Examples in Microsoft Visual J++](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ADO Events Model Example (VC++)

The Visual C++ section of [ADO Event Instantiation by Language](#) gives a general description of how to instantiate the ADO event model. The following is a specific example of instantiating the event model within the environment created by the **#import** directive.

The general description uses **adoint.h** as a reference for method signatures. However, a few details in the general description change slightly as a result of using the **#import** directive:

- The **#import** directive resolves **typedef**'s, and method signature data types and modifiers to their fundamental forms.
- The pure virtual methods that must be overwritten are all prefixed by "**raw_**".

Some of the code simply reflects coding style.

- The pointer to **IUnknown** used by the **Advise** method is obtained explicitly with a call to **QueryInterface**.
- You don't need to explicitly code a destructor in the class definitions.
- You may want to code more robust implementations of **QueryInterface**, **AddRef**, and **Release**.
- The **__uuidof()** directive is used extensively to obtain interface IDs.

Finally, the example contains some working code.

- The example is written as a console application.
- You should insert your own code under the comment, "**// Do some work**".
- All the event handlers default to doing nothing, and canceling further notifications. You should insert the appropriate code for your application, and allow notifications if required.

```
// eventmodel.cpp : Defines the entry point for the console applicat  
//
```

```
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \  
    no_namespace rename("EOF", "EndOfFile")
```

```

#include <comdef.h>
#include <stdio.h>

//----The Connection events-----

class CConnEvent : public ConnectionEventsVt
{
private:
    ULONG    m_cRef;
public:
    CConnEvent() { m_cRef = 0; };
    ~CConnEvent() {};

    STDMETHODCALLTYPE QueryInterface(REFIID riid, void **ppv);
    STDMETHODCALLTYPE AddRef(void);
    STDMETHODCALLTYPE Release(void);

    STDMETHODCALLTYPE raw_InfoMessage(
        struct Error *pError,
        EventStatusEnum *adStatus,
        struct _Connection *pConnection);

    STDMETHODCALLTYPE raw_BeginTransComplete(
        LONG TransactionLevel,
        struct Error *pError,
        EventStatusEnum *adStatus,
        struct _Connection *pConnection);

    STDMETHODCALLTYPE raw_CommitTransComplete(
        struct Error *pError,
        EventStatusEnum *adStatus,
        struct _Connection *pConnection);

    STDMETHODCALLTYPE raw_RollbackTransComplete(
        struct Error *pError,
        EventStatusEnum *adStatus,
        struct _Connection *pConnection);

    STDMETHODCALLTYPE raw_WillExecute(
        BSTR *Source,
        CursorTypeEnum *CursorType,
        LockTypeEnum *LockType,
        long *Options,
        EventStatusEnum *adStatus,
        struct _Command *pCommand,
        struct _Recordset *pRecordset,
        struct _Connection *pConnection);

    STDMETHODCALLTYPE raw_ExecuteComplete(

```

```

        LONG RecordsAffected,
        struct Error *pError,
        EventStatusEnum *adStatus,
        struct _Command *pCommand,
        struct _Recordset *pRecordset,
        struct _Connection *pConnection);

STDMETHODIMP raw_WillConnect(
    BSTR *ConnectionString,
    BSTR *UserID,
    BSTR *Password,
    long *Options,
    EventStatusEnum *adStatus,
    struct _Connection *pConnection);

STDMETHODIMP raw_ConnectComplete(
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Connection *pConnection);

STDMETHODIMP raw_Disconnect(
    EventStatusEnum *adStatus,
    struct _Connection *pConnection);
};

//-----The Recordset events-----

class CRstEvent : public RecordsetEventsVt
{
private:
    ULONG m_cRef;
public:
    CRstEvent() { m_cRef = 0; };
    ~CRstEvent() {};

    STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
    STDMETHODIMP_(ULONG) AddRef(void);
    STDMETHODIMP_(ULONG) Release(void);

    STDMETHODIMP raw_WillChangeField(
        LONG cFields,
        VARIANT Fields,
        EventStatusEnum *adStatus,
        struct _Recordset *pRecordset);

    STDMETHODIMP raw_FieldChangeComplete(
        LONG cFields,
        VARIANT Fields,
        struct Error *pError,

```

```

        EventStatusEnum *adStatus,
        struct _Recordset *pRecordset);

STDMETHODIMP raw_WillChangeRecord(
    EventReasonEnum adReason,
    LONG cRecords,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);

STDMETHODIMP raw_RecordChangeComplete(
    EventReasonEnum adReason,
    LONG cRecords,
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);

STDMETHODIMP raw_WillChangeRecordset(
    EventReasonEnum adReason,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);

STDMETHODIMP raw_RecordsetChangeComplete(
    EventReasonEnum adReason,
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);

STDMETHODIMP raw_WillMove(
    EventReasonEnum adReason,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);

STDMETHODIMP raw_MoveComplete(
    EventReasonEnum adReason,
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);

STDMETHODIMP raw_EndOfRecordset(
    VARIANT_BOOL *fMoreData,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);

STDMETHODIMP raw_FetchProgress(
    long Progress,
    long MaxProgress,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);

STDMETHODIMP raw_FetchComplete(

```

```

    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset);
};

//-----Implement each connection method-----

STDMETHODIMP CConnEvent::raw_InfoMessage(
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Connection *pConnection)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
};

STDMETHODIMP CConnEvent::raw_BeginTransComplete(
    LONG TransactionLevel,
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Connection *pConnection)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
};

STDMETHODIMP CConnEvent::raw_CommitTransComplete(
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Connection *pConnection)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
};

STDMETHODIMP CConnEvent::raw_RollbackTransComplete(
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Connection *pConnection)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
};

STDMETHODIMP CConnEvent::raw_WillExecute(
    BSTR *Source,
    CursorTypeEnum *CursorType,
    LockTypeEnum *LockType,

```

```
long *Options,  
EventStatusEnum *adStatus,  
struct _Command *pCommand,  
struct _Recordset *pRecordset,  
struct _Connection *pConnection)  
{  
*adStatus = adStatusUnwantedEvent;  
return S_OK;  
};
```

```
STDMETHODIMP CConnEvent::raw_ExecuteComplete(  
LONG RecordsAffected,  
struct Error *pError,  
EventStatusEnum *adStatus,  
struct _Command *pCommand,  
struct _Recordset *pRecordset,  
struct _Connection *pConnection)  
{  
*adStatus = adStatusUnwantedEvent;  
return S_OK;  
};
```

```
STDMETHODIMP CConnEvent::raw_WillConnect(  
BSTR *ConnectionString,  
BSTR *UserID,  
BSTR *Password,  
long *Options,  
EventStatusEnum *adStatus,  
struct _Connection *pConnection)  
{  
*adStatus = adStatusUnwantedEvent;  
return S_OK;  
};
```

```
STDMETHODIMP CConnEvent::raw_ConnectComplete(  
struct Error *pError,  
EventStatusEnum *adStatus,  
struct _Connection *pConnection)  
{  
*adStatus = adStatusUnwantedEvent;  
return S_OK;  
};
```

```
STDMETHODIMP CConnEvent::raw_Disconnect(  
EventStatusEnum *adStatus,  
struct _Connection *pConnection)  
{  
*adStatus = adStatusUnwantedEvent;  
return S_OK;  
};
```

```
//-----Implement each recordset method-----
```

```
STDMETHODIMP CRstEvent::raw_WillChangeField(
    LONG cFields,
    VARIANT Fields,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
};

STDMETHODIMP CRstEvent::raw_FieldChangeComplete(
    LONG cFields,
    VARIANT Fields,
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
};

STDMETHODIMP CRstEvent::raw_WillChangeRecord(
    EventReasonEnum adReason,
    LONG cRecords,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
};

STDMETHODIMP CRstEvent::raw_RecordChangeComplete(
    EventReasonEnum adReason,
    LONG cRecords,
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
{
    *adStatus = adStatusUnwantedEvent;
    return S_OK;
};

STDMETHODIMP CRstEvent::raw_WillChangeRecordset(
    EventReasonEnum adReason,
    EventStatusEnum *adStatus,
```

```

    struct _Recordset *pRecordset)
    {
        *adStatus = adStatusUnwantedEvent;
        return S_OK;
    };

STDMETHODIMP CRstEvent::raw_RecordsetChangeComplete(
    EventReasonEnum adReason,
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
    {
        *adStatus = adStatusUnwantedEvent;
        return S_OK;
    };

STDMETHODIMP CRstEvent::raw_WillMove(
    EventReasonEnum adReason,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
    {
        *adStatus = adStatusUnwantedEvent;
        return S_OK;
    };

STDMETHODIMP CRstEvent::raw_MoveComplete(
    EventReasonEnum adReason,
    struct Error *pError,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
    {
        *adStatus = adStatusUnwantedEvent;
        return S_OK;
    };

STDMETHODIMP CRstEvent::raw_EndOfRecordset(
    VARIANT_BOOL *fMoreData,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
    {
        *adStatus = adStatusUnwantedEvent;
        return S_OK;
    };

STDMETHODIMP CRstEvent::raw_FetchProgress(
    long Progress,
    long MaxProgress,
    EventStatusEnum *adStatus,
    struct _Recordset *pRecordset)
    {

```

```

        *adStatus = adStatusUnwantedEvent;
        return S_OK;
    };

    STDMETHODCALLTYPE CRstEvent::raw_FetchComplete(
        struct Error *pError,
        EventStatusEnum *adStatus,
        struct _Recordset *pRecordset)
    {
        *adStatus = adStatusUnwantedEvent;
        return S_OK;
    };

//-----Implement QueryInterface, AddRef, and Release-----

    STDMETHODCALLTYPE CRstEvent::QueryInterface(REFIID riid, void **ppv)
    {
        *ppv = NULL;
        if (riid == __uuidof(IUnknown) ||
            riid == __uuidof(RecordsetEventsVt)) *ppv = this;
        if (*ppv == NULL)
            return ResultFromCode(E_NOINTERFACE);
        AddRef();
        return NOERROR;
    }
    STDMETHODCALLTYPE CRstEvent::AddRef(void) { return ++m_cRef; }
    STDMETHODCALLTYPE CRstEvent::Release()
    {
        if (0 != --m_cRef) return m_cRef;
        delete this;
        return 0;
    }

    STDMETHODCALLTYPE CConnEvent::QueryInterface(REFIID riid, void **ppv)
    {
        *ppv = NULL;
        if (riid == __uuidof(IUnknown) ||
            riid == __uuidof(ConnectionEventsVt)) *ppv = this;
        if (*ppv == NULL)
            return ResultFromCode(E_NOINTERFACE);
        AddRef();
        return NOERROR;
    }
    STDMETHODCALLTYPE CConnEvent::AddRef() { return ++m_cRef; };
    STDMETHODCALLTYPE CConnEvent::Release()
    {
        if (0 != --m_cRef) return m_cRef;

```

```
        delete this;
        return 0;
    }
```

```
//-----Write your main block of code-----
```

```
int main(int argc, char* argv[])
{
    HRESULT hr;
    DWORD    dwConnEvt;
    DWORD    dwRstEvt;
    IConnectionPointContainer *pCPC = NULL;
    IConnectionPoint          *pCP = NULL;
    IUnknown                  *pUnk = NULL;
    CRstEvent                 *pRstEvent = NULL;
    CConnEvent                *pConnEvent= NULL;
    int                        rc = 0;
    _RecordsetPtr             pRst;
    _ConnectionPtr            pConn;

    ::CoInitialize(NULL);

    hr = pConn.CreateInstance(__uuidof(Connection));
    if (FAILED(hr)) return rc;

    hr = pRst.CreateInstance(__uuidof(Recordset));
    if (FAILED(hr)) return rc;

    // Start using the Connection events

    hr = pConn->QueryInterface(__uuidof(IConnectionPointContainer),
        (void **)&pCPC);
    if (FAILED(hr)) return rc;
    hr = pCPC->FindConnectionPoint(__uuidof(ConnectionEvents), &pCP);
    pCPC->Release();
    if (FAILED(hr)) return rc;

    pConnEvent = new CConnEvent();
    hr = pConnEvent->QueryInterface(__uuidof(IUnknown), (void **) &pU
    if (FAILED(hr)) return rc;
    hr = pCP->Advise(pUnk, &dwConnEvt);
    pCP->Release();
    if (FAILED(hr)) return rc;

    // Start using the Recordset events

    hr = pRst->QueryInterface(__uuidof(IConnectionPointContainer),
        (void **)&pCPC);
    if (FAILED(hr)) return rc;
    hr = pCPC->FindConnectionPoint(__uuidof(RecordsetEvents), &pCP);
```

```

pCPC->Release();
if (FAILED(hr)) return rc;

pRstEvent = new CRstEvent();
hr = pRstEvent->QueryInterface(__uuidof(IUnknown), (void **) &pUn
if (FAILED(hr)) return rc;
hr = pCP->Advise(pUnk, &dwRstEvt);
pCP->Release();
if (FAILED(hr)) return rc;

// Do some work

pConn->Open("dsn=Pubs;", "MyUserName", "MyPassword", adConnectUns
pRst->Open("SELECT * FROM authors", (IDispatch *) pConn,
          adOpenStatic, adLockReadOnly, adCmdText);
pRst->MoveFirst();
while (pRst->EndOfFile == FALSE)
{
    wprintf(L"Name = '%s'\n", (wchar_t*)
            ((_bstr_t) pRst->Fields->GetItem("au_lname")->Value));
    pRst->MoveNext();
}

pRst->Close();
pConn->Close();

// Stop using the Connection events

hr = pConn->QueryInterface(__uuidof(IConnectionPointContainer),
    (void **) &pCPC);
if (FAILED(hr)) return rc;
hr = pCPC->FindConnectionPoint(__uuidof(ConnectionEvents), &pCP);
pCPC->Release();
if (FAILED(hr)) return rc;
hr = pCP->Unadvise( dwConnEvt );
pCP->Release();
if (FAILED(hr)) return rc;

// Stop using the Recordset events
hr = pRst->QueryInterface(__uuidof(IConnectionPointContainer),
    (void **) &pCPC);
if (FAILED(hr)) return rc;
hr = pCPC->FindConnectionPoint(__uuidof(RecordsetEvents), &pCP);
pCPC->Release();
if (FAILED(hr)) return rc;
hr = pCP->Unadvise( dwRstEvt );
pCP->Release();
if (FAILED(hr)) return rc;

```

```
CoUninitialize();  
return 1;  
}
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

{

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
_RecordsetPtr  pRstEmployees  = NULL;

//Define Other Variables
//Interface Pointer declared.(VC++ Extensions)
IADORecordBinding  *picRs = NULL;
CEmployeeRs emprs;          //C++ class object
HRESULT hr = S_OK;
_bstr_t strMessage;

//Open a recordset using a Client Cursor
//For the Employee Table

_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

try
{
    // Open a recordset.
    TESTHR(hr = pRstEmployees.CreateInstance(__uuidof(Recordset))

    // Use client cursor to enable AbsolutePosition property.
    pRstEmployees->CursorLocation = adUseClient;

    // You have to explicitly pass the default Cursor type
    // and LockType to the Recordset here
    TESTHR(hr = pRstEmployees->Open("employee",
        strCnn, adOpenForwardOnly, adLockReadOnly, adCmdTable));

    //Open an IADORecordBinding interface pointer which we'll us
    //Binding Recordset to a class
    TESTHR(hr = pRstEmployees->QueryInterface
        (__uuidof(IADORecordBinding), (LPVOID*)&picRs));

    //Bind the Recordset to a C++ Class here
    TESTHR(hr = picRs->BindToRecordset(&empRs));

    //Display Names and hire dates, five records at a time
    pRstEmployees->PageSize = 5;

    int intPageCount = pRstEmployees->PageCount;

    for(int intPage=1;intPage<=intPageCount;intPage++)
    {
        pRstEmployees->put_AbsolutePage((enum PositionEnum)intPa

```

```

    strMessage = "";

    for(int intRecord=1;
        intRecord<=pRstEmployees->PageSize;intRecord++)
    {
        printf("\t%s %s %.10s\n",
            emprs.lau_fnameStatus == adFldOK ?
            emprs.m_szau_fname : "<NULL>",
            emprs.lau_lnameStatus == adFldOK ?
            emprs.m_szau_lname : "<NULL>",
            emprs.lau_hiredateStatus == adFldOK ?
            emprs.m_szau_hiredate : "<NULL>");

        pRstEmployees->MoveNext();

        if(pRstEmployees->EndOfFile)
            break;
    }

    printf("\n Press any key to continue...");getch();

    //Clear the Screen for the next Display
    system("cls");
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect = pRstEmployees->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            printf("Error:\n");
            printf("Code = %08lx\n", e.Error());
            printf("Message = %s\n", e.ErrorMessage());
            printf("Source = %s\n", (LPCSTR) e.Source());
            printf("Description = %s\n", (LPCSTR) e.Description());
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}
}

```

```

// Clean up objects before exit.
//Release the IADOResultset Interface here
if (picRs)
    picRs->Release();

if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
}

////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        printf("Error:\n");
        for(long iError = 0; iError < nCount; iError++)
        {
            pErr = pConnection->Errors->GetItem(iError);
            printf("\t Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}
//EndAbsolutePageCpp

```

AbsolutePageX.h

```

// BeginAbsolutePageH
#include "icrsint.h"

//This Class extracts only fname,lastname and hire_date

class CEmployeeRs : public CADOResultBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

```

```
//Column fname is the 2nd field in the table
ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szau_fname,
    sizeof(m_szau_fname), lau_fnameStatus, FALSE)

ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_szau_lname,
    sizeof(m_szau_lname), lau_lnameStatus, TRUE)

ADO_VARIABLE_LENGTH_ENTRY2(8, adVarChar, m_szau_hiredate,
    sizeof(m_szau_hiredate), lau_hiredateStatus, TRUE)

END_ADO_BINDING()

public:
    CHAR    m_szau_lname[41];
    ULONG   lau_lnameStatus;
    CHAR    m_szau_fname[41];
    ULONG   lau_fnameStatus;
    CHAR    m_szau_hiredate[40];
    ULONG   lau_hiredateStatus;

};
// EndAbsolutePageH
```

See Also

[AbsolutePage Property](#) | [PageCount Property](#) | [PageSize Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AbsolutePosition and CursorLocation Properties Example (VC++)

This example demonstrates how the [AbsolutePosition](#) property can track the progress of a loop that enumerates all the records of a [Recordset](#). It uses the [CursorLocation](#) property to enable the **AbsolutePosition** property by setting the [cursor](#) to a [client](#) cursor.

```
// BeginAbsolutePositionCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include "conio.h"
#include "AbsolutePositionX.h"

//Function Declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void AbsolutePositionX(void);
void AbsolutePosition2X(void);
void PrintProviderError(_ConnectionPtr pConnection);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    AbsolutePositionX();

    //Clear the screen for the next display
    printf("Press any key to continue...");
    getch();
    system("cls");

    AbsolutePosition2X();
}
```

```

        ::CoUninitialize();
    }

    ////////////////////////////////////////
    //                                     //
    //      AbsolutePositionX Function      //
    //                                     //
    ////////////////////////////////////////

void AbsolutePositionX(void)
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _RecordsetPtr pRstEmployees = NULL;

    //Define Other Variables
    //Interface Pointer declared.(VC++ Extensions)
    IADORecordBinding *picRs = NULL;
    CEmployeeRs emprs;           //C++ class object
    _bstr_t strMessage;
    char chKey;

    //Open a recordset using a Client Cursor
    //For the Employee Table

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        //Open a recordset.
        TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));

        //Use client cursor to enable AbsolutePosition property.
        pRstEmployees->CursorLocation = adUseClient;

        //You have to explicitly pass the default Cursor type
        //and LockType to the Recordset.
        TESTHR( pRstEmployees->Open("employee",
            strCnn, adOpenForwardOnly, adLockReadOnly, adCmdTable));

        // Open an IADORecordBinding interface pointer which we'll u
        // for Binding Recordset to a class.
        TESTHR(pRstEmployees->QueryInterface(
            __uuidof(IADORecordBinding), (LPVOID*)&picRs));

        //Bind the Recordset to a C++ Class here
    }
}

```

```

TESTHR(picRs->BindToRecordset(&emprs));

strMessage= "";

//Enumerate recordset
do
{
    //Display Current Record Information
    printf("Employee : %s \n record %ld of %d",
        emprs.lau_lnameStatus == adFldOK ?
        emprs.m_szau_lname : "<NULL>",
        pRstEmployees->AbsolutePosition,
        pRstEmployees->RecordCount);

    printf("\nContinue?(y/n)  :");

    do
    {
        chKey = getch();
    }while(chKey != 'y' && chKey !='n');

    //Clear the Screen for the next output
    system("cls");

    if(chKey == 'n')
        break;

    strMessage = "";
    pRstEmployees->MoveNext();
}while(!(pRstEmployees->EndOfFile));
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect = pRstEmployees->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            printf("Error:\n");
            printf("Code = %08lx\n", e.Error());
            printf("Message = %s\n", e.ErrorMessage());
            printf("Source = %s\n", (LPCSTR) e.Source());
            printf("Description = %s\n", (LPCSTR) e.Description());
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
    }
}

```

```

                break;
            default:
                printf("Errors occurred.");
                break;
        }
    }
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      AbsolutePosition2X Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void AbsolutePosition2X(void)
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _RecordsetPtr pRstEmployees = NULL;

    //Define Other Variables
    //Interface Pointer declared.(VC++ Extensions)
    IADORecordBinding *picRs = NULL;
    CEmployeeRs emprs;           //C++ class object
    _bstr_t strMessage;

    //Open a recordset using a Client Cursor
    //For the Employee Table

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        //Open a recordset.
        TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));
    }
}

```

```

//Use client cursor to enable AbsolutePosition property.
pRstEmployees->CursorLocation = adUseClient;

//You have to explicitly pass the default Cursor type
//and LockType to the Recordset.
TESTHR(pRstEmployees->Open("employee",
    strConn,adOpenStatic,adLockReadOnly,adCmdTable));

// Open an IADORecordBinding interface pointer which we'll u
// for Binding Recordset to a class.
TESTHR(pRstEmployees->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

//Bind the Recordset to a C++ Class here
TESTHR(picRs->BindToRecordset(&emprs));

long lGoToPos = 21;

pRstEmployees->AbsolutePosition = (PositionEnum)lGoToPos;

//Display Current Record Information
printf("Employee : %s \n record %ld of %d",
    emprs.lau_lnameStatus == adFldOK ? emprs.m_szau_lname :
    pRstEmployees->RecordCount);

printf("\nPress any key to continue:");
getch();
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect = pRstEmployees->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            printf("Error:\n");
            printf("Code = %08lx\n", e.Error());
            printf("Message = %s\n", e.ErrorMessage());
            printf("Source = %s\n", (LPCSTR) e.Source());
            printf("Description = %s\n", (LPCSTR) e.Description());
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
    }
}

```

```

        break;
    }
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr    = NULL;
    long        nCount  = 0;
    long        i        = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s",
                pErr->Number, (LPCSTR) pErr->Description);
        }
    }
}

// EndAbsolutePositionCpp

```

AbsolutePositionX.h

```

// BeginAbsolutePositionH
#include <ole2.h>

```

```
#include <stdio.h>
#include "icrsint.h"

//This Class extracts lastname.

class CEmployeeRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

    //Column lname is the 4th field in the recordset

    ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_szau_lname,
        sizeof(m_szau_lname), lau_lnameStatus, TRUE)

END_ADO_BINDING()

public:
    CHAR    m_szau_lname[41];
    ULONG   lau_lnameStatus;

};
// EndAbsolutePositionH
```

See Also

[AbsolutePosition Property](#) | [CursorLocation Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActiveCommand Property Example (VC++)

This example demonstrates the [ActiveCommand](#) property.

A subroutine is given a [Recordset](#) object whose **ActiveCommand** property is used to display the command text and parameter that created the **Recordset**.

```
// BeginActiveCommandCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include "ActiveCommandX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void ActiveCommandX(void);
void ActiveCommandXprint(_RecordsetPtr pRst);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);
inline char* mygets(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return NULL;

    if (!strchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strchr(strDest, '\n') - strDest] = '\0';

    return pstrRet;
}
```

```

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//          Main Function                                             //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    ActiveCommandX();

    ::CoUninitialize();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//          ActiveCommandX Function                                   //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void ActiveCommandX(void)
{
    HRESULT    hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _ConnectionPtr pConnection    = NULL;
    _CommandPtr    pCmd           = NULL;
    _RecordsetPtr pRstAuthors    = NULL;

    //Definitions of other variables
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");
    _bstr_t strPrompt;
    _bstr_t strName;
    CHAR strcharName[50];

    try
    {
        // Open connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));

        TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));
    }
}

```

```

TESTHR(pCmd.CreateInstance(__uuidof(Command)));

printf ("ActiveCommandX Example\n\n");
strPrompt = "Enter an author's name (e.g., Ringer): ";
printf(strPrompt);
mygets(strcharName, 50);
char *tempStr = strtok(strcharName, " \t");
strName = tempStr;

pCmd->CommandText = "SELECT * FROM authors WHERE au_lname =
pCmd->Parameters->Append(pCmd->CreateParameter("LastName", a

pConnection->Open (strCnn, "", "", adConnectUnspecified);

pCmd->PutActiveConnection(_variant_t((IDispatch*)pConnection

pRstAuthors = pCmd->Execute(NULL, NULL, adCmdText);

ActiveCommandXprint(pRstAuthors);
} // End Try statement.
catch(_com_error &e)
{
// Notify the user of errors if any.
// Pass a connection pointer accessed from the Recordset.
PrintProviderError(pConnection);
PrintComError(e);
}

// Clean up objects before exit.
if (pRstAuthors)
if (pRstAuthors->State == adStateOpen)
pRstAuthors->Close();
if (pConnection)
if (pConnection->State == adStateOpen)
pConnection->Close();
}

////////////////////////////////////
//
// ActiveCommandXprint Function //
//
////////////////////////////////////

void ActiveCommandXprint(_RecordsetPtr pRst = NULL)
{
// Variable Declaraion & initialization

```

```

IADORecordBinding *picRs = NULL; //Interface Pointer declare
CAuthorsRs autrs; //C++ class object
_bstr_t strName;

//Open an IADORecordBinding interface pointer which
//we'll use for Binding Recordset to a class
TESTHR(pRst->QueryInterface(__uuidof(IADORecordBinding), (LPVOID*)

//Bind the Recordset to a C++ Class here
TESTHR(picRs->BindToRecordset(&autrs));

strName = ((_CommandPtr)pRst->GetActiveCommand())->GetParameters
printf("Command text = '%s'\n", (LPCSTR)((_CommandPtr)pRst->GetA
printf("Parameter = '%s'\n", (LPCSTR)strName);

if (pRst->BOF)
    printf("Name = '%s'not found.", (LPCSTR)strName);
else
{
    printf ("Name = '%s %s' author ID = '%s'",
            autrs.lau_fnameStatus == adFldOK ? autrs.m_au_fname : "<
            autrs.lau_lnameStatus == adFldOK ? autrs.m_au_lname : "<
            autrs.lau_idStatus == adFldOK ? autrs.m_au_id : "<NULL>"
}

//Release IADORecordset Interface
if (picRs)
    picRs->Release();
}

////////////////////////////////////
//                                                                    //
//          PrintProviderError Function                               //
//                                                                    //
////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;
    long        nCount = 0;
    long        i      = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)

```

```

        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s\n", pErr->Number, (LPCST
        }
    }
}

```

```

////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////

```

```

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndActiveCommandCpp

```

```

ActiveCommandX.h
// BeginActiveCommandH
#include "icrsint.h"

```

```

// This Class extracts id, fname, lname from authors table.

```

```

class CAuthorsRs : public CADOResultBinding
{
BEGIN_ADO_BINDING(CAuthorsRs)

    // Column au_id is the 1st field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_au_id,
        sizeof(m_au_id), lau_idStatus, TRUE)

    // Column au_fname is the 2nd field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_au_fname,
        sizeof(m_au_fname), lau_fnameStatus, TRUE)

    // Column au_lname is the 3rd field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_au_lname,

```

```
        sizeof(m_au_lname), lau_lnameStatus, TRUE)
END_ADO_BINDING()
public:
    char    m_au_id[21];
    ULONG   lau_idStatus;
    char    m_au_fname[41];
    ULONG   lau_fnameStatus;
    char    m_au_lname[41];
    ULONG   lau_lnameStatus;
};
// EndActiveCommandH
```

See Also

[ActiveCommand Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example (VC++)

This example uses the [ActiveConnection](#), [CommandText](#), [CommandTimeout](#), [CommandType](#), [Size](#), and [Direction](#) properties to execute a stored procedure.

```
// BeginActiveConnectionCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#define TESTHR(x) if FAILED(x) _com_issue_error(x)

#include <stdio.h>
#include <ole2.h>
#include "conio.h"
#include "ActiveConnectionX.h"

//Function declaration
void ActiveConnectionX(VOID);
void PrintProviderError(_ConnectionPtr pConnection);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    ActiveConnectionX();

    //Wait here for user to see the output..
    printf("\n\nPress any key to continue..");
    getch();

    ::CoUninitialize();
}
```

```
}
```

```
////////////////////////////////////  
//                                                                    //  
//      ActiveConnectionX Function                                    //  
//                                                                    //  
////////////////////////////////////  
VOID ActiveConnectionX(VOID)  
{  
    HRESULT hr = S_OK;  
  
    // Define ADO object pointers.  
    // Initialize pointers on define.  
    // These are in the ADO:: namespace.  
    _ConnectionPtr pConnection      = NULL;  
    _CommandPtr    pCmdByRoyalty    = NULL;  
    _RecordsetPtr pRstByRoyalty     = NULL;  
    _RecordsetPtr pRstAuthors       = NULL;  
    _ParameterPtr pPrmByRoyalty     = NULL;  
  
    //Define Other variables  
    IADORecordBinding *picRs = NULL; //Interface Pointer declared  
    CEmployeeRs emprs;           //C++ class object TCS(SPA)  
    _bstr_t strAuthorId;  
    int intRoyalty;  
    VARIANT vtroyal ;  
  
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"  
                  "Initial Catalog='pubs';Integrated Security='SSPI'");  
  
    try  
    {  
        //Define a command object for a stored procedure.  
  
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));  
        hr = pConnection->Open(strCnn, "", "", adConnectUnspecified);  
  
        TESTHR(pCmdByRoyalty.CreateInstance(__uuidof(Command)));  
  
        pCmdByRoyalty->ActiveConnection = pConnection;  
        pCmdByRoyalty->CommandText = "byRoyalty";  
        pCmdByRoyalty->CommandType = adCmdStoredProc;  
        pCmdByRoyalty->CommandTimeout = 15;  
  
        //Define stored procedure's input parameter.  
        printf("Enter Royalty : ");  
        scanf("%d",&intRoyalty);  
    }  
}
```

```

//Assign Integer value
vtroyal.vt = VT_I2;
vtroyal.iVal = intRoyalty;

TESTHR(pPrmByRoyalty.CreateInstance(__uuidof(Parameter)));
pPrmByRoyalty->Type = adInteger;
pPrmByRoyalty->Size = 3;
pPrmByRoyalty->Direction = adParamInput;
pPrmByRoyalty->Value = vtroyal;
pCmdByRoyalty->Parameters->Append(pPrmByRoyalty);

//Create a recordset by executing a command.
pRstByRoyalty = pCmdByRoyalty->Execute(NULL, NULL, adCmdStored

//Open the authors table to get author names for display.

TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));
hr = pRstAuthors->Open("authors", strCnn, adOpenForwardOnly, ad

//Open an IADORecordBinding interface pointer which we'll us
TESTHR(pRstAuthors->QueryInterface(__uuidof(IADORecordBindin

//Bind the Recordset to a C++ Class here
TESTHR(picRs->BindToRecordset(&emprs));

//Print current data in the recordset ,adding author names f
printf("Authors With %d Percent Royalty",intRoyalty);

while(!(pRstByRoyalty->EndOfFile))
{
    strAuthorId = pRstByRoyalty->Fields->Item["au_id"]->Valu
    pRstAuthors->Filter = "au_id = '"+strAuthorId+"'";

    printf("\n\t%s, %s %s", emprs.lau_idStatus == adFldOK ? e
    emprs.lau_fnameStatus == adFldOK ? emprs.m_szau_fname :
        emprs.lau_lnameStatus == adFldOK ? emprs.m_szau_

    pRstByRoyalty->MoveNext();
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    PrintProviderError(pConnection);
}

```

```

        printf("Source : %s \n Description : %s \n", (LPCSTR)bstrSou
    }

    // Clean up objects before exit.
    //Release the IAD0Recordset Interface here
    if (picRs)
        picRs->Release();

    if (pRstByRoyalty)
        if (pRstByRoyalty->State == adStateOpen)
            pRstByRoyalty->Close();
    if (pRstAuthors)
        if (pRstAuthors->State == adStateOpen)
            pRstAuthors->Close();
    if (pConnection)
        if (pConnection->State == adStateOpen)
            pConnection->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VOID PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;
    long        nCount = 0;
    long        i      = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s", pErr->Number, (LPCSTR)pErr
        }
    }
}

// EndActiveConnectionCpp

```

ActiveConnectionX.h

```
// BeginActiveConnectionH
#include "icrsint.h"

//This Class extracts fname,lastname

class CEmployeeRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

    //Column au_id is the 1st field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_szau_id,
        sizeof(m_szau_id), lau_idStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szau_lname,
        sizeof(m_szau_lname), lau_lnameStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_szau_fname,
        sizeof(m_szau_fname), lau_fnameStatus, TRUE)

END_ADO_BINDING()

public:

    CHAR m_szau_id[20];
    ULONG lau_idStatus;

    CHAR m_szau_fname[40];
    ULONG lau_fnameStatus;

    CHAR m_szau_lname[40];
    ULONG lau_lnameStatus;

};
// EndActiveConnectionH
```

See Also

[ActiveConnection Property](#) | [CommandText Property](#) | [CommandTimeout Property](#) | [CommandType Property](#) | [Direction Property](#) | [Size Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActualSize and DefinedSize Properties Example (VC++)

This example uses the [ActualSize](#) and [DefinedSize](#) properties to display the defined size and actual size of a field.

```
// BeginActualSizeCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void ActualSizeX(VOID);
void PrintProviderError(_ConnectionPtr pConnection);

/////////////////////////////////////////////////////////////////
//                                                                    //
//      Main Function                                                    //
//                                                                    //
/////////////////////////////////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;
    ActualSizeX();
    ::CoUninitialize();
}

/////////////////////////////////////////////////////////////////
//                                                                    //
//      ActualSizeX Function                                             //
//                                                                    //
/////////////////////////////////////////////////////////////////
VOID ActualSizeX(VOID)
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
```

```

_RecordsetPtr pRstStores = NULL;

//Define Other variables
_bstr_t strMessage;
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
    "Initial Catalog='pubs';Integrated Security='SSPI'");

try
{
    //Open a recordset for the stores table.
    TESTHR(pRstStores.CreateInstance(__uuidof(Recordset)));

    //You have to explicitly pass the Cursor type and LockType
    //to the Recordset here.
    hr = pRstStores->Open("stores",
        strCnn,adOpenForwardOnly,adLockReadOnly,adCmdTable);

    //Loop through the recordset displaying the contents
    //of the stor_name field, the field's defined size,
    //and its actual size.

    pRstStores->MoveFirst();

    while(!(pRstStores->EndOfFile))
    {
        strMessage = "Store Name: ";
        strMessage += (_bstr_t)pRstStores->Fields->
            Item["stor_name"]->Value + "\n";
        strMessage += "Defined Size: ";
        strMessage += (_bstr_t)pRstStores->Fields->
            Item["stor_name"]->DefinedSize + "\n";
        strMessage += "Actual Size: ";
        strMessage += (_bstr_t) pRstStores->Fields->
            Item["stor_name"]->ActualSize + "\n";

        printf("%s\n", (LPCSTR)strMessage);
        printf("Press any key to continue...");
        getch();
        //Clear the screen for the next display
        system("cls");
        pRstStores->MoveNext();
    }
}
catch(_com_error &e)
{
    _variant_t vtConnect = pRstStores->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)

```

```

    {
        case VT_BSTR:
            printf("Error:\n");
            printf("Code = %08lx\n", e.Error());
            printf("Message = %s\n", e.ErrorMessage());
            printf("Source = %s\n", (LPCSTR) e.Source());
            printf("Description = %s\n", (LPCSTR) e.Description());
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
if (pRstStores)
    if (pRstStores->State == adStateOpen)
        pRstStores->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VOID PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;
    long        nCount = 0;
    long        i     = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number, (LPCSTR) )
        }
    }
}
// EndActualSizeCpp

```

See Also

[ActualSize Property](#) | [DefinedSize Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AddNew Method Example (VC++)

This example uses the [AddNew](#) method to create a new record with the specified name.

```
// Note: When adding the record. You need to get the data from the u
// The employee id must be formatted as first,middle and last initia
// five numbers,then M or F to signify the gender.For example,the
// employee id for Bill A. Sorensen would be "BAS55555M".
```

```
// BeginAddNewCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")
```

```
#include <ole2.h>
#include <stdio.h>
#include "conio.h"
#include "AddNewX.h"
```

```
//Function declaration
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void AddNewX(VOID);
void PrintProviderError(_ConnectionPtr pConnection);
inline int myscanf(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return 0;

    if (!strchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strchr(strDest, '\n') - strDest] = '\0';

    return strlen(strDest);
}
```

```

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
void main()
{
    HRESULT hr = S_OK;

    if(FAILED(::CoInitialize(NULL)))
        return;

    if (SUCCEEDED(hr))
    {
        AddNewX();

        //Wait here for the user to see the output
        printf("Press any key to continue...");
        getch();

        ::CoUninitialize();
    }
}

////////////////////////////////////
//                                                                    //
//      AddNewX Function                                              //
//                                                                    //
////////////////////////////////////
VOID AddNewX(VOID)
{

    // Define ADO object pointers.
    // Initialize pointers on define.
    _RecordsetPtr pRstEmployees = NULL;
    _ConnectionPtr pConnection = NULL;

    //Define Other variables
    IADORecordBinding *picRs = NULL; //Interface Pointer declared.(
    CEmployeeRs emprs; //C++ class object

    HRESULT hr = S_OK;

                                //Replace Data Source value with your se
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");
    _bstr_t strId;
    _bstr_t strMessage;

```

```

try
{
    //Open a connection
    TESTHR(pConnection.CreateInstance(__uuidof(Connection));
    pConnection->Open(strCnn, "", "", adConnectUnspecified);

    //Open employee table
    TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));

    //You have to explicitly pass the Cursor type and LockType t
    pRstEmployees->Open("employee",_variant_t((IDispatch *) pCon
        adOpenKeyset,adLockOptimistic,adCmdTable);

    //Open an IADORecordBinding interface pointer which we'll us
    //Recordset to a class
    TESTHR(pRstEmployees->QueryInterface(__uuidof(IADORecordBind

    //Bind the Recordset to a C++ Class here
    TESTHR(picRs->BindToRecordset(&emprs));

    // Get data from the user.The employee id must be formatted
    // first,middle and last initial,five numbers,then M or F to
    // signify the gender.For example,the employee id for
    // Bill A. Sorensen would be "BAS55555M".

    printf("Enter Employee Id: ");
    myscanf(emprs.m_sz_empid, sizeof(emprs.m_sz_empid));
    strId = emprs.m_sz_empid;
    printf("Enter First Name: ");
    myscanf(emprs.m_sz_fname, sizeof(emprs.m_sz_fname));
    printf("Enter Last Name:");
    myscanf(emprs.m_sz_lname, sizeof(emprs.m_sz_lname));

    //Proceed if the user actually entered some thing
    //for the id, the first and the last name.

    if(strcmp(emprs.m_sz_empid,"") && strcmp(emprs.m_sz_fname,""
        strcmp(emprs.m_sz_lname,""))
    {
        //This adds a new record to the table
        //if (FAILED(hr = picRs->AddNew(&emprs)))
        //_com_issue_error(hr);
        TESTHR(picRs->AddNew(&emprs));

        //Show the newly added data
        printf("New Record: %s %s %s \n",\
            emprs.lemp_empidStatus == adFldOK ? emprs.m_sz_empid : "

```

```

        emprs.lemp_fnameStatus == adFldOK ? emprs.m_sz_fname : "
        emprs.lemp_lnameStatus == adFldOK ? emprs.m_sz_lname : "
    }
    else
        printf("Please enter an employee id, first name and last

//Delete the new record because this is a demonstration.
pConnection->Execute("DELETE FROM EMPLOYEE WHERE emp_id = "
    NULL,adCmdText);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect = pRstEmployees->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            printf("Error:\n");
            printf("Code = %08lx\n", e.Error());
            printf("Message = %s\n", e.ErrorMessage());
            printf("Source = %s\n", (LPCSTR) e.Source());
            printf("Description = %s\n", (LPCSTR) e.Description());
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
//Release the IADOResultset Interface here
if (picRs)
    picRs->Release();

if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

```

```

////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////

VOID PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr  pErr = NULL;
    long      nCount = 0;
    long      i = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\n\t Error number: %x\t%s", pErr->Number, (LPCST
        }
    }
}
// EndAddNewCpp

```

AddNewX.h

```

// BeginAddNewH
#include "icrsint.h"

//This Class extracts empid, fname and lastname

class CEmployeeRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

    //Column empid is the 1st field in the recordset

    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_sz_empid,
        sizeof(m_sz_empid), lemp_empidStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_sz_fname,
        sizeof(m_sz_fname), lemp_fnameStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_sz_lname,
        sizeof(m_sz_lname), lemp_lnameStatus, TRUE)

```

```
END_ADO_BINDING()  
  
public:  
  
    CHAR    m_sz_empid[10];  
    ULONG   lemp_empidStatus;  
    CHAR    m_sz_fname[40];  
    ULONG   lemp_fnameStatus;  
    CHAR    m_sz_lname[41];  
    ULONG   lemp_lnameStatus;  
  
};  
// EndAddNewH
```

See Also

[AddNew Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
VOID AppendX(VOID)
{

    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr pRstByRoyalty = NULL;
    _RecordsetPtr pRstAuthors = NULL;
    _CommandPtr pcmdByRoyalty = NULL;
    _ParameterPtr pprmByRoyalty = NULL;
    _ConnectionPtr pConnection = NULL;

    //Define Other variables
    IADORecordBinding *picRs = NULL; //Interface Pointer declared
    CEmployeeRs emprs; //C++ class object

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    _bstr_t strMessage, strAuthorID;

    int intRoyalty;
    VARIANT vtRoyalty;

    try
    {
        //Open a Connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        hr = pConnection->Open(strCnn, "", "", adConnectUnspecified);
        pConnection->CursorLocation = adUseClient;

        //Open Command Object with one Parameter
        TESTHR(pcmdByRoyalty.CreateInstance(__uuidof(Command));
        pcmdByRoyalty->CommandText = "byroyalty";
        pcmdByRoyalty->CommandType = adCmdStoredProc;

        //Get parameter value and append parameter
        printf("Enter Royalty: ");
        scanf("%d",&intRoyalty);

        //Define Integer/variant.
        vtRoyalty.vt = VT_I2;
        vtRoyalty.iVal = intRoyalty;
        pprmByRoyalty = pcmdByRoyalty->CreateParameter("percentage",
        pcmdByRoyalty->Parameters->Append(pprmByRoyalty);

```

```

pprmByRoyalty->Value = vtRoyalty;

//Create Recordset by executing the command
pcmdByRoyalty->ActiveConnection = pConnection;
pRstByRoyalty = pcmdByRoyalty->Execute(NULL, NULL, adCmdStored

//Open the authors table to get author names for display
TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));

//You have to explicitly pass the default Cursor type and Lo
hr = pRstAuthors->Open("authors", _variant_t((IDispatch*)pCon

//Open an IADORecordBinding interface pointer which we'll us
TESTHR(pRstAuthors->QueryInterface(__uuidof(IADORecordBindin

//Bind the Recordset to a C++ Class here
TESTHR(picRs->BindToRecordset(&emprs));

//Print current data in the recordset, adding
//author names from author table.
printf("Authors with %d percent royalty ", intRoyalty);

while(!(pRstByRoyalty->EndOfFile))
{
    strAuthorID = pRstByRoyalty->Fields->Item["au_id"]->Valu
    pRstAuthors->Filter = "au_id = '"+strAuthorID+"'";

    printf("\n" "%s, %s %s", emprs.lau_idStatus == adFldOK
        emprs.lau_fnameStatus == adFldOK ? emprs.m_szau_
        emprs.lau_lnameStatus == adFldOK ? emprs.m_szau_

        pRstByRoyalty->MoveNext();
}
}
catch(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    PrintProviderError(pConnection);

    printf("\n Source : %s \n Description : %s \n", (LPCSTR)bstrS
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

```

```

    if (pRstByRoyalty)
        if (pRstByRoyalty->State == adStateOpen)
            pRstByRoyalty->Close();
    if (pRstAuthors)
        if (pRstAuthors->State == adStateOpen)
            pRstAuthors->Close();
    if (pConnection)
        if (pConnection->State == adStateOpen)
            pConnection->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VOID PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;
    long        nCount = 0;
    long        i = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\n Error Description: %s\n", pEr
        }
    }
}
// EndAppendCpp

```

AppendX.h

```

// BeginAppendH
#include "icrsint.h"

//This Class extracts only author id, fname, lastname
class CEmployeeRs : public CADORecordBinding

```

```

{
BEGIN_ADO_BINDING(CEmployeeRs)

    //Column au_id is the 1st field in the recordset

    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_szau_id,
        sizeof(m_szau_id), lau_idStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szau_lname,
        sizeof(m_szau_lname), lau_lnameStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_szau_fname,
        sizeof(m_szau_fname), lau_fnameStatus, TRUE)

END_ADO_BINDING()

public:

    CHAR m_szau_id[20];
    ULONG lau_idStatus;

    CHAR m_szau_fname[40];
    ULONG lau_fnameStatus;

    CHAR m_szau_lname[40];
    ULONG lau_lnameStatus;

};
// EndAppendH

```

See Also

[Append Method](#) | [CreateParameter Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AppendChunk and GetChunk Methods Example (VC++)

This example uses the [AppendChunk](#) and [GetChunk](#) methods to fill an image field with data from another record.

```
// BeginAppendChunkCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#define ChunkSize    100

#include <ole2.h>
#include <stdio.h>
#include "conio.h"
#include "malloc.h"
#include "AppendChunkX.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void AppendChunkX(VOID);
void PrintProviderError(_ConnectionPtr pConnection);
inline int myscanf(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return 0;

    if (!strrchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strrchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strrchr(strDest, '\n') - strDest] = '\0';

    return strlen(strDest);
}
inline char* mygets(char* strDest, int n)
{
```

```

char strExBuff[10];
char* pstrRet = fgets(strDest, n, stdin);

if (pstrRet == NULL)
    return NULL;

if (!strrchr(strDest, '\n'))
    // Exhaust the input buffer.
    do
    {
        fgets(strExBuff, sizeof(strExBuff), stdin);
    }while (!strrchr(strExBuff, '\n'));
else
    // Replace '\n' with '\0'
    strDest[strrchr(strDest, '\n') - strDest] = '\0';

return pstrRet;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      Main Function                                                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void main()
{
    HRESULT hr = S_OK;

    if(FAILED(::CoInitialize(NULL)))
        return;

    AppendChunkX();

    //Wait here for the user to see the output
    printf("\n\nPress any key to continue..");
    getch();
    ::CoUninitialize();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      AppendChunkX Function                                                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
VOID AppendChunkX(VOID)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.

```

```

_RecordsetPtr pRstPubInfo = NULL;
_ConnectionPtr pConnection = NULL;

//Define other variables
IADORecordBinding *picRs = NULL; //Interface Pointer declared
CPubInfoRs pubrs; //C++ class object

HRESULT hr = S_OK;
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
    "Initial Catalog='pubs';Integrated Security='SSPI'");

_bstr_t strMessage, strPubID, strPRInfo;
_variant_t varChunk;
long lngOffset, lngLogoSize;
char pubId[50];
lngOffset = 0;

UCHAR chData;
SAFEARRAY FAR *psa;
SAFEARRAYBOUND rgsabound[1];
rgsabound[0].lLbound = 0;
rgsabound[0].cElements = ChunkSize;

try
{
    //Open a Connection.
    TESTHR(pConnection.CreateInstance(__uuidof(Connection));
    hr = pConnection->Open(strCnn, "", "", adConnectUnspecified);

    TESTHR(hr= pRstPubInfo.CreateInstance(__uuidof(Recordset)));

    pRstPubInfo->CursorType = adOpenKeyset;
    pRstPubInfo->LockType = adLockOptimistic;

    hr = pRstPubInfo->Open("pub_info",
        _variant_t((IDispatch*)pConnection, true),
        adOpenKeyset, adLockOptimistic, adCmdTable);

    //Open an IADORecordBinding interface pointer which we'll us
    //for Binding Recordset to a class
    TESTHR(pRstPubInfo->QueryInterface(
        __uuidof(IADORecordBinding), (LPVOID*)&picRs));

    //Bind the Recordset to a C++ Class here
    TESTHR(picRs->BindToRecordset(&pubrs));

    //Display the available logos here
    strMessage = "Available logos are: " + (_bstr_t)"\n\n";
    printf(strMessage);
}

```

```

int Counter = 0;
while(!(pRstPubInfo->EndOfFile))
{
    printf("\n%s",pubrs.m_sz_pubid);
    printf("\n%s",strtok(pubrs.m_sz_prinfo","));

    //Display 5 records at a time and wait for user to conti
    if (++Counter >= 5)
    {
        Counter = 0;
        printf("\nPress any key to continue...");
        getch();
    }
    pRstPubInfo->MoveNext();
}

//Prompt For a Logo to Copy
printf("\nEnter the ID of a logo to copy: ");
myscanf(pubId, sizeof(pubId));
strPubID = pubId;

//Copy the logo to a variable in chunks

pRstPubInfo->Filter = "pub_id = '" + strPubID + "'";
lngLogoSize = pRstPubInfo->Fields->Item["logo"]->ActualSize;

//Create a safe array to store the array of BYTES
rgsabound[0].cElements = lngLogoSize;
psa = SafeArrayCreate(VT_UI1,1,rgsabound);

long index1 = 0;
while(lngOffset < lngLogoSize)
{
    varChunk = pRstPubInfo->Fields->
                Item["logo"]->GetChunk(ChunkSize);

    //Copy the data only upto the Actual Size of Field.
    for(long index=0;index<=(ChunkSize-1);index++)
    {
        hr= SafeArrayGetElement(varChunk.parray,&index,&chDa
        if(SUCCEEDED(hr))
        {
            //Take BYTE by BYTE and advance Memory Location
            hr = SafeArrayPutElement(psa,&index1,&chData);
            index1++;
        }
        else
            break;
    }
    lngOffset = lngOffset + ChunkSize;
}

```

```

}
lngOffset = 0;

printf("Enter a new Pub Id: ");
myscanf(pubrs.m_sz_pubid, sizeof(pubrs.m_sz_pubid));
strPubID = pubrs.m_sz_pubid;
printf("Enter descriptive text: " );
mygets(pubrs.m_sz_prinfo, sizeof(pubrs.m_sz_prinfo));

pRstPubInfo->AddNew();
pRstPubInfo->Fields->GetItem("pub_id")->PutValue(strPubID);
pRstPubInfo->Fields->GetItem("pr_info")->
    PutValue(pubrs.m_sz_prinfo);

//Assign the Safe array to a variant.
varChunk.vt = VT_ARRAY|VT_UI1;
varChunk.parray = psa;
hr = pRstPubInfo->Fields->GetItem("logo")->
    AppendChunk(varChunk);

//Update the table
pRstPubInfo->Update();

lngLogoSize = pRstPubInfo->Fields->Item["logo"]->ActualSize;

//Show the newly added record.
printf("New Record : %s\n Description : %s\n Logo Size : %s"
        pubrs.m_sz_pubid,
        pubrs.m_sz_prinfo, (LPCSTR)(_bstr_t)pRstPubInfo->Fields->
        Item["logo"]->ActualSize);

//Delete new record because this is demonstration.
pConnection->Execute("DELETE FROM PUB_INFO WHERE pub_id = '"
                    + strPubID + "'", NULL, adCmdText);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    PrintProviderError(pConnection);
    printf("Source : %s \n Description : %s\n", (LPCSTR)bstrSource
        (LPCSTR)bstrDescription);
}

// Clean up objects before exit.
if (pRstPubInfo)
    if (pRstPubInfo->State == adStateOpen)

```

```

        pRstPubInfo->Close();
    if (pConnection)
        if (pConnection->State == adStateOpen)
            pConnection->Close();
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VOID PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;
    long        nCount = 0;
    long        i = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,(LPCSTR)
        }
    }
}

// EndAppendChunkCpp

```

AppendChunkX.h

```

// BeginAppendChunkH
#include "icrsint.h"

//This Class extracts pubid,prinfo.

class CPubInfoRs : public CADORecordBinding
{
    BEGIN_ADO_BINDING(CPubInfoRs)

        ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_sz_pubid,
            sizeof(m_sz_pubid), l_pubid, TRUE)

```

```
        ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_sz_prinfo,  
        sizeof(m_sz_prinfo), l_prinfo, TRUE)  
  
    END_ADO_BINDING()  
  
public:  
    CHAR    m_sz_pubid[10];  
    ULONG   l_pubid;  
    CHAR    m_sz_prinfo[200];  
    ULONG   l_prinfo;  
};  
// EndAppendChunkH
```

See Also

[AppendChunk Method](#) | [Field Object](#) | [GetChunk Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

//                                                                    //
//      AttributesX Function                                          //
//                                                                    //
////////////////////////////////////
void AttributesX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _RecordsetPtr pRstEmployee = NULL;
    _ConnectionPtr pConnection = NULL;
    FieldsPtr fldLoop = NULL;
    PropertiesPtr proLoop = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;
    _variant_t Index;
    Index.vt = VT_I2;
    int j=0;
    //Open a recordset using a Client Cursor
    //For the Employee Table
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
    "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // open connection and record set
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        pConnection->Open(strCnn, "", "", adConnectUnspecified);

        TESTHR(pRstEmployee.CreateInstance(__uuidof(Recordset));
        pRstEmployee->Open("Employee", _variant_t((IDispatch *)pConn
        adLockReadOnly, adCmdTable);

        // Display the attributes of Connection.
        printf("Connection attributes: %d \n", pConnection->Attribut

        // Display the attribute of the employee table's
        //fields
        printf("\nFields attributes:\n");
        fldLoop = pRstEmployee->GetFields();

        for (int i = 0; i < (int)fldLoop->GetCount(); i++)
        {
            Index.iVal=i;
            printf ("    %s = %d \n", (LPSTR)fldLoop->GetItem(Index)->
            (int)fldLoop->GetItem(Index)->GetAttributes());
        }
    }
}

```

```

// Display Fields of the Employee table which are NULLBALE
printf("\nNULLABLE Fields :");

for (int i1 = 0; i1 < (int)fldLoop->GetCount(); i1++)
{
    Index.iVal = i1;

    if (fldLoop->GetItem(Index)->GetAttributes() & adFldIsN
        {
            printf ("%s \n", (LPSTR)fldLoop->GetItem(Index)
        }
}

// Display the attributes of the Employee tables's
// properties
printf("\nProperty attributes:\n");
proLoop = pRstEmployee->GetProperties();

for (int i2 = 0; i2 < (int)proLoop->GetCount(); i2++)
{
    j= j+1;
    Index.iVal=i2;
    printf (" %s = %d \n", (LPSTR)(_bstr_t)proLoop->GetItem(
        ,(int)proLoop->GetItem(Index)->GetAttributes());

    if (((j % 23) == 0) || ( i2==6))
    {
        printf("\nPress any key to continue...");
        getch();

        //Clear the screen for the next display
        system("cls");
        j=0;
    }
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.

    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
if (pRstEmployee)
    if (pRstEmployee->State == adStateOpen)
        pRstEmployee->Close();

```

```

    if (pConnection)
        if (pConnection->State == adStateOpen)
            pConnection->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.

    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;
    long        nCount = 0;
    long        i     = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", (LPCSTR) pErr->Number,
                }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VOID PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("\nError\n");
    printf("Code = %08lx\n", e.Error());
    printf("Code meaning = %s\n", e.ErrorMessage());
    printf("Source = %s\n", (LPCSTR) bstrSource);
}

```

```
        printf("Description = %s\n", (LPCSTR) bstrDescription);
    }
// EndAttributesCpp
```

AttributesX.h

```
// BeginAttributesH
#include "icrsint.h"

//This class extracts LastName, FirstName, FaxPhone from Employees t
class CEmployeeRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

        // Column LastName is the 2nd field in the table
        ADO_VARIABLE_LENGTH_ENTRY2(2,adVarChar,m_szemp_LastName,
            sizeof(m_szemp_LastName),lemp_LastNameStatus,TRUE)

        // Column FirstName is the 17th field in the table
        ADO_VARIABLE_LENGTH_ENTRY2(17,adVarChar,m_szemp_FirstName,
            sizeof(m_szemp_FirstName),lemp_FirstNameStatus,TRUE)

        // Column FaxPhone is the 18th field in the table
        ADO_VARIABLE_LENGTH_ENTRY2(18,adVarChar,m_szemp_Faxphone,
            sizeof(m_szemp_Faxphone),lemp_FaxphoneStatus,TRUE)

END_ADO_BINDING()

public:
    CHAR   m_szemp_LastName[21];
    ULONG  lemp_LastNameStatus;
    CHAR   m_szemp_FirstName[11];
    ULONG  lemp_FirstNameStatus;
    CHAR   m_szemp_Faxphone[25];
    ULONG  lemp_FaxphoneStatus;
};

// EndAttributesH
```

See Also

[Attributes Property](#) | [Connection Object](#) | [Field Object](#) | [Name Property](#) | [Property Object](#)

ADO 2.5 Samples 

BeginTrans, CommitTrans, and RollbackTrans Methods Example (VC++)

This example changes the book type of all psychology books in the *Titles* table of the database. After the [BeginTrans](#) method starts a transaction that isolates all the changes made to the *Titles* table, the [CommitTrans](#) method saves the changes. You can use the [Rollback](#) method to undo changes that you saved using the [Update](#) method.

```
// BeginBeginTransCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <conio.h>
#include <assert.h>
#include <malloc.h>
#include "BeginTransX.h"

inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void BeginTransX(void);
void PrintProviderError(_ConnectionPtr pConnection);

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    BeginTransX();

    //Wait here for user to see the output.
    printf("\nPress any key to continue...");
    getch();

    ::CoUninitialize();
}

////////////////////////////////////
//
```

```

//      BeginTransX Function                                //
//                                                                 //
//////////////////////////////////////////////////////////////////

void BeginTransX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr rstTitles = NULL;
    _ConnectionPtr pConnection = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;
    IADORecordBinding *picRs = NULL; //Interface Pointer declared
    CTitlesRs titlrs;
    _bstr_t strTitle;
    _bstr_t strMessage;
    LPSTR p_TempStr = NULL;
    char chKey;
    int i = 0;

    try
    {
        // open connection.
        _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer'
            "Initial Catalog='pubs';Integrated Security='SSPI'");

        TESTHR(pConnection.CreateInstance(__uuidof(Connection));

        TESTHR(pConnection->Open(strCnn, "", "", adConnectUnspecified))

        rstTitles.CreateInstance(__uuidof(Recordset));

        rstTitles->CursorType = adOpenDynamic;
        rstTitles->LockType = adLockPessimistic;

        // open Titles table
        TESTHR(rstTitles->Open("titles",
            _variant_t((IDispatch*)pConnection, true),
            adOpenDynamic, adLockPessimistic, adCmdTable));

        rstTitles->MoveFirst();
        pConnection->BeginTrans();

        //Open an IADORecordBinding interface pointer which
        //we'll use for Binding Recordset to a class
        TESTHR(rstTitles->QueryInterface(
            __uuidof(IADORecordBinding), (LPVOID*)&picRs));
    }
}

```

```

//Bind the Recordset to a C++ Class here
TESTHR(picRs->BindToRecordset(&titlrs));

// Loop through recordset and ask user if he wants
// to change the type for a specified title.
// Allocate memory to p_TempStr string pointer.
p_TempStr = (LPSTR) malloc(sizeof(titlrs.m_szT_type));

// Check for null string.
assert(p_TempStr != NULL);
while (VARIANT_FALSE == rstTitles->EndOfFile)
{
    // Set the final character of the destination string to
    p_TempStr[sizeof(titlrs.m_szT_type)-1] = '\\0';
    // The source string will get truncated if its length is
    // longer than the length of the destination string minu
    strncpy(p_TempStr, strtok(titlrs.m_szT_type, " "), sizeof(t

    // Compare type with psychology
    if (!strcmp(p_TempStr, "psychology"))
    {
        strTitle = titlrs.m_szT_title;
        strMessage = "Title: " + strTitle +
            "\\n Change type to Self help?(y/n)";

        // Change the title for specified employee
        printf("%s\\n", (LPCSTR)strMessage);
        do
        {
            chKey = getch();
        }while(chKey != 'y' && chKey !='n');
        if(chKey == 'y')
        {
            // Set the final character of the destination st
            titlrs.m_szT_type[sizeof(titlrs.m_szT_type)-1] =
            // Copy "self_help" title field.
            // The source string will get truncated if its l
            // longer than the length of the destination str
            strncpy(titlrs.m_szT_type, "self_help", sizeof(ti
            picRs->Update(&titlrs);
        }
    }
    rstTitles->MoveNext();
}
// Ask if the User wants to commit to all the
// changes made above
printf("\\n\\n Save all changes(y/n)?");
do
{

```

```

        chKey = getch();
    }while(chKey != 'y' && chKey !='n');

    if(chKey == 'y')

        // Save the changes to the title table
        pConnection->CommitTrans();
    else
        // Unsave the changes to the title table
        pConnection->RollbackTrans();

    // Print current data in recordset.
    rstTitles->Requery(0);

    // Move to the first record of the title table
    rstTitles->MoveFirst();
    printf("\n\nPress any key to continue...");
    getch();

    // Clear the screen for the next display
    //system("cls");

    // Open IADORecordBinding interface pointer again
    // for binding Recordset to a class.
    TESTHR(rstTitles->QueryInterface(
        __uuidof(IADORecordBinding),
        (LPVOID*)&picRs));

    // Rebind the Recordset to a C++ Class.
    TESTHR(picRs->BindToRecordset(&titlrs));

    while (!rstTitles->EndOfFile)
    {
        i= i+1;
        if (i % 23 == 0)
        {
            printf("\nPress any key to continue...");
            getch();

            //Clear the screen for the next display
            //system("cls");
        }
        printf("%s - %s\n",titlrs.m_szT_title,titlrs.m_szT_type);
        rstTitles->MoveNext();
    }
    // Restore original data because this is a demonstration.
    rstTitles->MoveFirst();

    while (VARIANT_FALSE == rstTitles->EndOfFile)
    {

```

```

        // Set the final character of the destination string to
        p_TempStr[sizeof(titlrs.m_szT_type)-1] = '\\0';
        // The source string will get truncated if its length is
        // longer than the length of the destination string minu
        strncpy(p_TempStr,titlrs.m_szT_type,sizeof(titlrs.m_szT_
        p_TempStr = strtok(p_TempStr," ");

        if (!strcmp(p_TempStr,"self_help"))
        {
            // Set the final character of the destination string
            titlrs.m_szT_type[sizeof(titlrs.m_szT_type)-1] = '\\0
            // The source string will get truncated if its lengt
            // longer than the length of the destination string
            strncpy(titlrs.m_szT_type,"psychology",sizeof(titlrs
            picRs->Update(&titlrs);
        }
        rstTitles->MoveNext();
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    PrintProviderError(pConnection);

    printf("Source : %s\\n", (LPCSTR)bstrSource);
    printf("Description : %s\\n", (LPCSTR)bstrDescription);
}

// Deallocate the memory
if (p_TempStr)
    free(p_TempStr);
// Clean up objects before exit.
//Release the IAD0Recordset Interface here
if (picRs)
    picRs->Release();

if (rstTitles)
    if (rstTitles->State == adStateOpen)
        rstTitles->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
};

//////////////////////////////////////
//

```

```

//      PrintProviderError Function                                     //
//                                                                 //
//////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr  pErr    = NULL;
    long      nCount  = 0;
    long      i       = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number, (LPCSTR) p
        }
    }
}
// EndBeginTransCpp

```

BeginTransX.h

```

// BeginBeginTransH
#include "icrsint.h"

//This Class extracts only title and type
class CTitlesRs : public CADOResultBinding
{
BEGIN_ADO_BINDING(CTitlesRs)
    //Column title is the 2nd field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szT_title,
        sizeof(m_szT_title), lT_titleStatus, FALSE)

    //Column type is the 3rd field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_szT_type,
        sizeof(m_szT_type), lT_typeStatus, TRUE)
END_ADO_BINDING()

public:
    CHAR    m_szT_title[150];
    ULONG   lT_titleStatus;
    CHAR    m_szT_type[40];
    ULONG   lT_typeStatus;
};

```

```
// EndBeginTransH
```

See Also

[BeginTrans, CommitTrans, and RollbackTrans Methods](#) | [Update Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

BOF, EOF, and Bookmark Properties

Example (VC++)

The first function in this example uses the [BOF](#) and [EOF](#) properties to display a message if a user tries to move past the first or last record of a [Recordset](#). It uses the [Bookmark](#) property to let the user flag a record in a **Recordset** and return to it later.

The second function uses the **Bookmark** property to place the **Bookmark** of every other record from a **Recordset** into an array, and then filters the Recordset using the array.

```
// BeginBOFCpp
#import "C:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include "BofEofBookmark.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void BOFX(void);
void BookmarkX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                               //
//      BOFX Function            //
//                               //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    BOFX();

    //Clear the screen for the next display
```

```

system("cls");

BookmarkX();

printf("Press any key to continue...");
getch();

::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//      BOFX Function                                                //
//                                                                    //
////////////////////////////////////

void BOFX(void)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _RecordsetPtr rstPublishers = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;
    IADORecordBinding *picRs = NULL;    //Interface Pointer declare
    CPublishers Pubs;

    bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");
    _bstr_t strMessage;
    _variant_t VarBookmark;
    int intCommand = 0;
    _variant_t TempPublisher;

    try
    {
        // Open recordset with data from Publishers table.
        TESTHR(rstPublishers.CreateInstance(__uuidof(Recordset)));
        rstPublishers->CursorType = adOpenStatic;

        // Use client cursor to enable absolutePosition property.
        rstPublishers->CursorLocation = adUseClient;
        rstPublishers->Open("select pub_id, pub_name from publishers
            " order by pub_name", strCnn, adOpenStatic,
            adLockBatchOptimistic, adCmdText);

        //Open an IADORecordBinding interface pointer
        //which will be used for Binding Recordset to a class
        TESTHR(rstPublishers->QueryInterface(

```

```

    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

//Bind the Recordset to a C++ Class here
TESTHR(picRs->BindToRecordset(&Publs));

rstPublishers->MoveFirst();

while (true)    // Continuous loop.
{
    // Display information about the current record
    // and get user input
    printf("Publisher:%s \n Record %d of %d\n\n",
        Publs.lP_pubnameStatus == adFldOK ?
        Publs.m_szP_pubname : "<NULL>",
        rstPublishers->AbsolutePosition,
        rstPublishers->RecordCount);
    printf("Enter command:\n ");
    printf("[1 - next          / 2 - previous          /\n");
    printf(" 3 - set bookmark / 4 - go to bookmark /\n");
    printf(" 5 - quit                ]\n");

    scanf("%d", &intCommand);
    if ((intCommand < 1) || (intCommand > 4))
        break;    // Out of range entry exits program loop.

    switch(intCommand)
    {
        // Move forward or backward, trapping for BOF or EOF
        case 1:
            rstPublishers->MoveNext();
            if (rstPublishers->EndOfFile)
            {
                printf("\nCannot move past the last record."
                    " Try again...\n");
                rstPublishers->MoveLast();
            }
            break;

        case 2:
            rstPublishers->MovePrevious();
            if (rstPublishers->BOF)
            {
                printf("\nCannot move before the first recor
                    " Try again...\n");
                rstPublishers->MoveFirst();
            }
            break;

        // store the bookmark of the current record.

```

```

        case 3:
            VarBookmark = rstPublishers->Bookmark;
            // Go to the record indicated by the
            // stored bookmark
            break;

        case 4:
            // Check for whether bookmark set for a record
            if (VarBookmark.vt == VT_EMPTY)
                printf("No Bookmark set!\n");
            else
                rstPublishers->Bookmark = VarBookmark;
            break;

        default:
            break;
    }
}
}
catch (_com_error &e)
{
    printf("Error in BOFx...\n");
    // Notify the user of errors if any.
    _variant_t vtConnect = rstPublishers->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            printf("Error:\n");
            printf("Code = %08lx\n", e.Error());
            printf("Message = %s\n", e.ErrorMessage());
            printf("Source = %s\n", (LPCSTR) e.Source());
            printf("Description = %s\n", (LPCSTR) e.Description());
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
    printf("Press any key to continue...");
    getch();
}

// Clean up objects before exit.
//Release the IADORecordset Interface here

```

```

    if (picRs)
        picRs->Release();

    if (rstPublishers)
        if (rstPublishers->State == adStateOpen)
            rstPublishers->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      BookmarkX Function                                                                                   //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void BookmarkX(void)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr rstAuthors = NULL;

    //Define Other Variables
    IADORecordBinding *picRs = NULL; //Interface Pointer declared.
    CAuthors Authrs;
    HRESULT hr = S_OK;
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");
    _variant_t vBookmark;

    // Variable declaration for safe arrays.
    SAFEARRAY FAR* psa;

    // define ARRAY/ VARIANT variant.
    vBookmark.vt = VT_ARRAY|VT_VARIANT;
    SAFEARRAYBOUND rgsabound[1];
    rgsabound[0].lLbound = 0;
    rgsabound[0].cElements = 11;
    long ii = 0;

    try
    {
        rstAuthors.CreateInstance(__uuidof(Recordset));
        // Set The Cursor Location
        rstAuthors->CursorLocation = adUseClient;
        rstAuthors->PutActiveConnection((_variant_t)strCnn);

        // Open Authors table
        TESTHR(rstAuthors->Open("select * from authors",strCnn,
            adOpenStatic,adLockBatchOptimistic,adCmdText));
    }
}

```

```

//Open an IADORecordBinding interface pointer
//which we'll use for binding Recordset to a class
TESTHR(rstAuthors->QueryInterface(__uuidof(IADORecordBinding
(LPVOID*)&picRs));

//Bind the Recordset to a C++ Class here
TESTHR(picRs->BindToRecordset(&Authrs));

printf("Number of Records before filtering:  %d\n",
      rstAuthors->RecordCount);

// Create safearrays to store array of variant
psa = SafeArrayCreate(VT_VARIANT,1,rgsabound);

// Store bookmark of every other record into an array.
while ((!rstAuthors->EndOfFile) && (ii < 11))
{
    SafeArrayPutElement(psa,&ii,&rstAuthors->Bookmark);
    //ii = ii +1;
    ii++;
    rstAuthors->Move(2);
}

vBookmark.parray = psa;

// Filter the Record with the array of bookmarks.
rstAuthors->put_Filter(vBookmark);
printf("Number of Records after filtering:  %d\n",
      rstAuthors->RecordCount);
rstAuthors->MoveFirst();

while (!rstAuthors->EndOfFile)
{
    printf("%d    %s\n",rstAuthors->AbsolutePosition,
          Authrs.lau_lnameStatus == adFldOK ?
          Authrs.m_szau_lname : "<NULL>");
    rstAuthors->MoveNext();
}
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect = rstAuthors->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:

```

```

        printf("Error:\n");
        printf("Code      = %08lx\n", e.Error());
        printf("Message = %s\n", e.ErrorMessage());
        printf("Source  = %s\n", (LPCSTR) e.Source());
        printf("Description = %s\n", (LPCSTR) e.Description());
        break;
    case VT_DISPATCH:
        PrintProviderError(vtConnect);
        break;
    default:
        printf("Errors occurred.");
        break;
    }
}

// Clean up objects before exit.
//Release the IADORRecordset Interface here
if (picRs)
    picRs->Release();

if (rstAuthors)
    if (rstAuthors->State == adStateOpen)
        rstAuthors->Close();
}

////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr  pErr    = NULL;
    long      nCount  = 0;
    long      i       = 0;

    if( (pConnection->Errors->Count) > 0)
    {
        nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

```

```
    }  
}  
// EndBOFCpp
```

BofEOFBookmark.h

```
// BeginBOFEOFH  
#include "icrsint.h"  
  
//This Class extracts only pubid,lastname and hire_date  
class CPublishers : public CADORecordBinding  
{  
BEGIN_ADO_BINDING(CPublishers)  
  
    //Column title is the 2nd field in the recordset  
    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_szP_pubid,  
        sizeof(m_szP_pubid), lP_pubidStatus, FALSE)  
  
    //Column type is the 3rd field in the recordset  
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szP_pubname,  
        sizeof(m_szP_pubname), lP_pubnameStatus, TRUE)  
  
END_ADO_BINDING()  
  
public:  
    CHAR    m_szP_pubid;  
    ULONG   lP_pubidStatus;  
    CHAR    m_szP_pubname[40];  
    ULONG   lP_pubnameStatus;  
};  
  
//This Class extracts only authorlastname  
class CAuthors : public CADORecordBinding  
{  
BEGIN_ADO_BINDING(CAuthors)  
  
    //Column authorlname is the 2nd field in the recordset  
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szau_lname,  
        sizeof(m_szau_lname), lau_lnameStatus, FALSE)  
  
END_ADO_BINDING()  
  
public:  
    CHAR    m_szau_lname[40];  
    ULONG   lau_lnameStatus;  
};  
// EndBOFEOFH
```

See Also

[BOF, EOF Properties](#) | [Bookmark Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

void CacheSizeX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _RecordsetPtr    pRstRoySched = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;
    DWORD sngStart;
    DWORD sngEnd;
    float sngNoCache;
    float sngCache;
    int intLoop = 0;
    _bstr_t strTemp;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        //Open the RoySched table.
        TESTHR(pRstRoySched.CreateInstance(__uuidof(Recordset)));
        pRstRoySched->Open("roysched", strCnn, adOpenForwardOnly,
            adLockReadOnly, adCmdTable);

        // Enumerate the Recordset object twice and record
        // the elapsed time.
        sngStart = GetTickCount();

        for (intLoop = 1; intLoop < 2; intLoop++)
        {
            pRstRoySched->MoveFirst();

            while(!(pRstRoySched->EndOfFile))
            {
                // Execute a simple operation for the
                // performance test.
                strTemp = pRstRoySched->Fields->
                    Item["title_id"]->Value;
                pRstRoySched->MoveNext();
            }
        }
        sngEnd = GetTickCount();
        sngNoCache = (float)(sngEnd - sngStart)/(float)1000;

        // Cache records in groups of 30 records.
        pRstRoySched->MoveFirst();
        pRstRoySched->CacheSize = 30;
    }
}

```

```

sngStart = GetTickCount();

// Enumerate the Recordset object twice and record
// the elapsed time.
for (intLoop = 1;intLoop < 2; intLoop++)
{
    pRstRoySched->MoveFirst();
    while(!(pRstRoySched->EndOfFile))
    {
        // Execute a simple operation for the
        // performance test.
        strTemp = pRstRoySched->Fields->
            Item["title_id"]->Value;
        pRstRoySched->MoveNext();
    }
}
sngEnd = GetTickCount();
sngCache = (float)(sngEnd - sngStart)/(float)1000;

// Display performance results.
printf("Caching Performance Results:\n");
printf("No cache: %6.3f  seconds \n", sngNoCache);
printf("30-record cache: %6.3f  seconds \n", sngCache);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect = pRstRoySched->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            // Pass a connection pointer accessed from the Record
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
if (pRstRoySched)

```

```

        if (pRstRoySched->State == adStateOpen)
            pRstRoySched->Close();
    }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.

    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                       //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}

// EndCacheSizeCpp

```

See Also

[CacheSize Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

void CancelX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _ConnectionPtr pConnection = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;
    _bstr_t strCmdChange;
    _bstr_t strCmdRestore;
    BOOL booChanged = FALSE;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // open a connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        pConnection->Open(strCnn, "", "", adConnectUnspecified);

        // Define command strings.
        strCmdChange = "UPDATE titles SET type = 'self_help' "
            "WHERE type = 'psychology'";

        strCmdRestore = "UPDATE titles SET type = 'psychology' "
            "WHERE type = 'self_help'";

        // Begin a transaction, then execute a command asynchronousl
        pConnection->BeginTrans();
        pConnection->Execute(strCmdChange, NULL, adAsyncExecute);

        // do something else for a little while - this could be chan
        for (int i = 1; i<=100000 ;i++)
        {
            // i = i + i;
            printf("%d\n", i);
        }

        // If the command has NOT completed, cancel the execute
        // and roll back the transaction. Otherwise, commit the
        // transaction.
        if ((pConnection->GetState()))
        {
            pConnection->Cancel();
            pConnection->RollbackTrans();
            booChanged = FALSE;
            printf("Update canceled.\n");
            printf("GetState = %d\n", pConnection->GetState());
        }
    }
}

```

```

    }
    else
    {
        pConnection->CommitTrans();
        booChanged = TRUE;
        printf("Update complete.\n");
    }

    // If the change was made, restore the data
    // because this is a demonstration.
    if (booChanged)
    {
        pConnection->Execute(strCmdRestore, NULL, 0);
        printf("Data restored.\n");
    }
}
catch(_com_error &e)
{
    // Notify user of any errors that result from
    // executing the query.
    // Pass a connection pointer accessed from the Connection.
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Cleanup object before exit
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.

```

```

        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndCancelCpp

```

See Also

[Cancel Method](#) | [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Clone Method Example (VC++)

This example uses the [Clone](#) method to create copies of a [Recordset](#) and then lets the user position the record pointer of each copy independently.

```
// BeginCloneCpp
#import "C:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <conio.h>
#include "CloneX.h"

// Function Declarations.
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void CloneX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);
inline char* mygets(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return NULL;

    if (!strrchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strrchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strrchr(strDest, '\n') - strDest] = '\0';

    return pstrRet;
}

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
```

```

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    CloneX();

    //Wait here for user to see the output..
    printf("\nPress any key to continue...");
    getch();
    ::CoUninitialize();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      CloneX Function                                                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void CloneX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _RecordsetPtr arstStores[3];

    //Define Other Variables
    HRESULT hr = S_OK;
    int intLoop = 0;
    _bstr_t strSQL;
    _bstr_t strMessage;
    _bstr_t strFind;
    int intLoop1 = 0;
    char *tempStr;
    bool boolFlag = TRUE;
    char m_szS_stor_name[150];

    // Assign SQL statement and connection string to variables.
    strSQL = "SELECT stor_name FROM Stores ORDER BY stor_name";

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open recordset as a static cursor type recordset.
        arstStores[0].CreateInstance(__uuidof(Recordset));
        arstStores[0]->CursorType = adOpenStatic;
        arstStores[0]->LockType = adLockBatchOptimistic;
    }
}

```

```

TESTHR(arstStores[0]->Open(strSQL, strCnn, adOpenStatic,
    adLockBatchOptimistic, adCmdText));

// Create two clones of the original Recordset.
arstStores[1] = arstStores[0]->Clone(adLockUnspecified);
arstStores[2] = arstStores[0]->Clone(adLockUnspecified);

while (boolFlag)
{
    // Loop through the array so that on each pass,
    // the user is searching a different copy of the
    // same Recordset.
    for (intLoop = 1; intLoop <= 3 ; intLoop++)
    {
        // Ask for search string while showing where
        // the current record pointer is for each Recordset.
        printf("Recordsets from stores table:\n");

        _bstr_t str1 = arstStores[0]->Fields->
            GetItem("stor_name")->Value;
        printf("\t1 - Original - Record pointer at %s",
            (LPCSTR)str1);

        _bstr_t str2 = arstStores[1]->Fields->
            GetItem("stor_name")->Value;
        printf("\n\t2 - Clone - Record pointer at %s",
            (LPCSTR)str2);

        _bstr_t str3 = arstStores[2]->Fields->
            GetItem("stor_name")->Value;
        printf("\n\t3 - Clone - Record pointer at %s",
            (LPCSTR)str3);

        printf("\n\nEnter search string for # %d, "
            "or press Enter to quit.\n", intLoop);
        mygets(m_szS_stor_name, 150);

        // Trim the String.
        tempStr = strtok(m_szS_stor_name, " \t");
        strMessage = tempStr;
        if (tempStr == NULL)
        {
            boolFlag = FALSE;
            break;
        }

        // Find the search string; if there's no
        // match, jump to the last record.
        intLoop1 = intLoop - 1;
    }
}

```

```

        arstStores[intLoop1]->Filter = "stor_name >= '" +
            strMessage + "'";

        if (arstStores[intLoop1]->EndOfFile)
        {
            arstStores[intLoop1]->Filter = (long)adFilterNon
            arstStores[intLoop1]->MoveLast();
        }
    } // End of While Loop
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect;

    vtConnect = arstStores[0]->GetActiveConnection();

    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
if (arstStores[0])
    if (arstStores[0]->State == adStateOpen)
        arstStores[0]->Close();
if (arstStores[1])
    if (arstStores[1]->State == adStateOpen)
        arstStores[1]->Close();
if (arstStores[2])
    if (arstStores[2]->State == adStateOpen)
        arstStores[2]->Close();
}

////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////

```

```

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
};

```

```

////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////

```

```

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
};
// EndCloneCpp

```

CloneX.h

```

// BeginCloneH
#include "icrsint.h"

// This Class extracts only store name
// from "stores" table.
class CStores : public CADRecordBinding
{

```

```
BEGIN_ADO_BINDING(CStores)

    //Column stor_name is the 1st field in the recordset

    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_szS_stor_name,
        sizeof(m_szS_stor_name), lS_stor_nameStatus, FALSE)

END_ADO_BINDING()

public:
    CHAR    m_szS_stor_name[150];
    ULONG   lS_stor_nameStatus;
};

// EndCloneH
```

See Also

[Clone Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

CompareBookmarks Method Example (VC++)

This example demonstrates the [CompareBookmarks](#) method. The relative value of bookmarks is seldom needed unless a particular bookmark is somehow special.

Designate a random row of a [Recordset](#) derived from the *Authors* table as the target of a search. Then display the position of each row relative to that target.

```
// BeginCompareBookmarksCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void CompareBookMarksX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                               //
//      Main Function            //
//                               //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    CompareBookMarksX();

    printf("Press any key to continue...");
    getch();
    ::CoUninitialize();
}
```

```

////////////////////////////////////
//                                                                    //
//          CompareBookMarksX Function                               //
//                                                                    //
////////////////////////////////////

void CompareBookMarksX(void)
{
    HRESULT hr = S_OK;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr pRstAuthors = NULL;
    _variant_t vTarget;
    _bstr_t strAns;
    _bstr_t strTitle;
    strTitle = "CompareBookmarks Example";
    try
    {
        TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));

        pRstAuthors->Open("SELECT * FROM authors ORDER BY au_id", st
            adOpenStatic, adLockReadOnly, adCmdText);

        long count = pRstAuthors->RecordCount;
        printf("Rows in the Recordset = %d\n", count);
        if (count == 0)
            exit(1); //Exit if an empty recordset

        srand( (unsigned)time( NULL ) ); //Randomize

        count = int(rand() % (count-1)); //Get position between 1 an
        if(!count) {count++;};

        printf("Randomly chosen row position = %d\n", count);

        _variant_t vtBookmark = (short)adBookmarkFirst;

        pRstAuthors->Move(count,vtBookmark); //Move row to random po

        vTarget = pRstAuthors->Bookmark; //Remember the mystery r
        count = 0;
        long intLineCnt = 3;
        pRstAuthors->MoveFirst();

        while (!(pRstAuthors->EndOfFile)) //Loop through recor

```

```

{
    intLineCnt++;
    if (intLineCnt%20 == 0)
    {
        printf("\nPress any key to continue...\n");
        getch();
    }
    long result = pRstAuthors->CompareBookmarks(
        pRstAuthors->Bookmark, vTarget);

    if (result == adCompareNotEqual)
        printf("Row %d: Bookmarks are not equal. %d\n", count, result);
    else if (result == adCompareNotComparable)
        printf("Row %d: Bookmarks are not comparable.\n", count, result);
    else
    {
        switch(result)
        {
            case adCompareLessThan:
                strAns = "less than";
                break;
            case adCompareEqual:
                strAns = "equal to";
                break;
            case adCompareGreaterThan:
                strAns = "greater than";
                break;
            default:
                strAns = "in error comparing to";
                break;
        }
        printf ("Row position %d is %s the target.\n",
            count, (LPCSTR)strAns);
    }
    count++;
    pRstAuthors->MoveNext();
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstAuthors->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:

```

```

        PrintComError(e);
        break;
    case VT_DISPATCH:
        PrintProviderError(vtConnect);
        break;
    default:
        printf("Errors occurred.");
        break;
    }
}

// Clean up objects before exit.
if (pRstAuthors)
    if (pRstAuthors->State == adStateOpen)
        pRstAuthors->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//          PrintProviderError Function                                                         //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//          PrintComError Function                                                             //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{

```

```
_bstr_t bstrSource(e.Source());
_bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndCompareBookmarksCpp
```

See Also

[CompareBookmarks Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ConnectionString, ConnectionTimeout, and State Properties Example (VC++)

This example demonstrates different ways of using the [ConnectionString](#) property to open a [Connection](#) object. It also uses the [ConnectionTimeout](#) property to set a connection timeout period, and the [State](#) property to check the state of the connections. The GetState function is required for this procedure to run.

```
// BeingConnectionStringCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void ConnectionStringX();
_bstr_t GetState(int intState);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    ConnectionStringX();

    //Wait here for user to see the output..
    printf("\nPress any key to continue...");
    getch();
}
```

```

        ::CoUninitialize();
    }

////////////////////////////////////
//                                                                    //
//      ConnectionStringX Function                                    //
//                                                                    //
////////////////////////////////////

void ConnectionStringX()
{
    // Define Connection object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _ConnectionPtr pConnection1 = NULL;
    _ConnectionPtr pConnection2 = NULL;
    _ConnectionPtr pConnection3 = NULL;
    _ConnectionPtr pConnection4 = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;

    try
    {
        // Open a connection using OLE DB syntax.
        TESTHR(pConnection1.CreateInstance(__uuidof(Connection)));
        pConnection1->ConnectionString =
            "Provider='sqloledb';Data Source='MySqlServer';"
            "Initial Catalog='Pubs';Integrated Security='SSPI';";
        pConnection1->ConnectionTimeout = 30;
        pConnection1->Open("", "", "", adConnectUnspecified);
        printf("cnn1 state: %s\n",
            (LPCTSTR)GetState(pConnection1->State));

        // Open a connection using a DSN and ODBC tags.
        // It is assumed that you have create DSN 'Pubs' with a user
        // 'MyUserId' and password as 'MyPassword'.
        TESTHR(pConnection2.CreateInstance(__uuidof(Connection)));
        pConnection2->ConnectionString = "DSN=Pubs;UID=MyUserId;PWD="
        pConnection2->Open("", "", "", adConnectUnspecified);
        printf("cnn2 state: %s\n",
            (LPCTSTR)GetState(pConnection2->State));

        // Open a connection using a DSN and OLE DB tags.
        TESTHR(pConnection3.CreateInstance(__uuidof(Connection)));
        pConnection3->ConnectionString = "Data Source=Pubs;";
        pConnection3->Open("", "", "", adConnectUnspecified);
        printf("cnn3 state: %s\n",
            (LPCTSTR)GetState(pConnection3->State));
    }
}

```

```

    // Open a connection using a DSN and individual
    // arguments instead of a connection string.
    // It is assumed that you have create DSN 'Pubs' with a user
    // 'MyUserId' and password as 'MyPassword'.
    TESTHR(pConnection4.CreateInstance(__uuidof(Connection));
    pConnection4->Open("Pubs", "MyUserId", "MyPassword", adConnectU
    printf("cnn4 state: %s\n",
        (LPCTSTR)GetState(pConnection4->State));
}
catch(_com_error &e)
{
    // Notify user of any errors.
    // Pass a connection pointer accessed from the Connection.
    PrintProviderError(pConnection1);
    if(pConnection2)
        PrintProviderError(pConnection2);
    if(pConnection3)
        PrintProviderError(pConnection3);
    if(pConnection4)
        PrintProviderError(pConnection4);
    PrintComError(e);
}

//Cleanup objects before exit.
if (pConnection1)
    if (pConnection1->State == adStateOpen)
        pConnection1->Close();
if (pConnection2)
    if (pConnection2->State == adStateOpen)
        pConnection2->Close();
if (pConnection3)
    if (pConnection3->State == adStateOpen)
        pConnection3->Close();
if (pConnection4)
    if (pConnection4->State == adStateOpen)
        pConnection4->Close();
}

////////////////////////////////////
//                                                                    //
//      GetState Function                                            //
//                                                                    //
////////////////////////////////////

_bstr_t GetState(int intState)
{
    _bstr_t strState;
    switch(intState)

```

```

    {
        case adStateClosed:
            strState = "adStateClosed";
            break;
        case adStateOpen:
            strState = "adStateOpen";
            break;
        default:
            ;
    }
    return strState;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                 //
//      PrintProviderError Function                                //
//                                                                 //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr  pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR)pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                 //
//      PrintComError Function                                    //
//                                                                 //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());
}

```

```
// Print Com errors.
printf("Error\n");
printf("\tCode = %08lx\n", e.Error());
printf("\tCode meaning = %s\n", e.ErrorMessage());
printf("\tSource = %s\n", (LPCSTR) bstrSource);
printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndConnectionStringCpp
```

See Also

[Connection Object](#) | [ConnectionString Property](#) | [ConnectionTimeout Property](#) | [State Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Count Property Example (VC++)

This example demonstrates the [Count](#) property with two collections in the *Employee* database. The property obtains the number of objects in each collection, and sets the upper limit for loops that enumerate these collections. Another way to enumerate these collections without using the **Count** property would be to use For Each...Next statements.

```
// BeginCountCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void CountX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//
//      Main Function
//
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    CountX();

    //Wait here for user to see the output..
    printf("\nPress any key to Exit...");
    getch();

    ::CoUninitialize();
}

////////////////////////////////////
//
```

```

//      CountX Function                                     //
//                                                                 //
/////////////////////////////////////////////////////////////////

void CountX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _RecordsetPtr pRstEmployee = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;
    _variant_t Index;
    Index.vt = VT_I2;
    int j = 0;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open recordset with data from Employee table.
        TESTHR(pRstEmployee.CreateInstance(__uuidof(Recordset)));
        pRstEmployee->Open("Employee", strCnn, adOpenForwardOnly,
            adLockReadOnly, adCmdTable);

        // Print information about Fields collection.
        printf("%d Fields in Employee\n", pRstEmployee->Fields->Count);
        for (int intLoop = 0;
            intLoop <= (pRstEmployee->Fields->Count-1);
            intLoop++)
        {
            Index.iVal = intLoop;
            printf(" %s\n", (LPSTR) pRstEmployee->Fields->
                GetItem(Index)->Name);
        }

        // Print information about Properties collection.
        printf("\n%d Properties in Employee\n", pRstEmployee->
            Properties->Count);
        for (intLoop = 0;
            intLoop <= (pRstEmployee->Properties->Count - 1);
            intLoop++)
        {
            j++;
            Index.iVal = intLoop;
            printf(" %s\n", (LPSTR)pRstEmployee->Properties->
                GetItem(Index)->Name);
            if (((j % 23) == 0) || (intLoop == 11))

```

```

        {
            printf("\nPress any key to continue...");
            getch();

            //Clear the screen for the next display
            system("cls");
            j = 0;
        }
    }
}
catch(_com_error &e)
{
    // Notify user of any errors that result from
    // executing the query.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstEmployee->GetActiveConnection();

    switch(vtConnect.vt)
    {
    case VT_BSTR:
        PrintComError(e);
        break;
    case VT_DISPATCH:
        PrintProviderError(vtConnect);
        break;
    default:
        printf("Errors occurred.");
        break;
    }
}
// Clean up objects before exit.
if (pRstEmployee)
    if (pRstEmployee->State == adStateOpen)
        pRstEmployee->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)

```

```

    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndCountCpp

```

See Also

[Count Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

CursorType, LockType, and EditMode Properties Example (VC++)

This example demonstrates setting the [CursorType](#) and [LockType](#) properties before opening a [Recordset](#). It also shows the value of the [EditMode](#) property under various conditions. The EditModeOutput function is required for this procedure to run.

```
// BeginEditModeCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declaration
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void EditModeX(void);
void EditModeOutput(char *, int);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    EditModeX();

    // Wait here for the user to see the output
    printf("\n\nPress any key to continue...");
    getch();

    ::CoUninitialize();
}
```

```

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      EditModeX Function                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void EditModeX(void)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr      pRstEmployees = NULL;
    _ConnectionPtr     pConnection   = NULL;

    HRESULT hr = S_OK;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open a connection
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        hr = pConnection->Open(strCnn, "", "", adConnectUnspecified);

        // Open recordset with data from employee table
        TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));

        pRstEmployees->CursorLocation = adUseClient;
        pRstEmployees->CursorType = adOpenStatic;
        pRstEmployees->LockType = adLockBatchOptimistic;

        pRstEmployees->Open("employee",
            _variant_t((IDispatch *) pConnection, true),
            adOpenStatic, adLockBatchOptimistic, adCmdTable);

        // Show the EditMode property under different editing states
        pRstEmployees->AddNew ();
        pRstEmployees->Fields->Item["emp_id"]->Value =
            (_bstr_t)("T-T55555M");
        pRstEmployees->Fields->Item["fname"]->Value =
            (_bstr_t)("temp_fname");
        pRstEmployees->Fields->Item["lname"]->Value =
            (_bstr_t)("temp_lname");
        EditModeOutput("After AddNew: ", pRstEmployees->EditMode);

        pRstEmployees->UpdateBatch(adAffectCurrent);
        EditModeOutput("After Update: ", pRstEmployees->EditMode);
    }
}

```

```

    pRstEmployees->Fields->Item["fname"]->Value = (_bstr_t)("tes
    EditModeOutput("After Edit: ", pRstEmployees->EditMode);

    // Delete new record because this is a demonstration.
    pConnection->Execute("DELETE FROM employee WHERE emp_id = "
        "'T-T55555M'", NULL, adCmdText);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

////////////////////////////////////
//                                                                    //
//      EditModeOutput() Function                                     //
//                                                                    //
////////////////////////////////////
void EditModeOutput(char *strTemp, int intEditMode)
{
    // Print report based on the value of the EditMode property.
    printf("\n%s", strTemp);
    printf("\n  EditMode = ");

    switch (intEditMode)
    {
        case adEditNone :
            printf("adEditNone");
            break;
        case adEditInProgress :
            printf("adEditInProgress");
            break;
        case adEditAdd :
            printf("adEditAdd");
            break;
    }
}
}

```

```

////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr  pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\n\t Error number: %x\t%s\n", pErr->Number,
                (LPCSTR)pErr->Description);
        }
    }
}

////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}

// EndEditModeCpp

```

See Also

[CursorType Property](#) | [EditMode Property](#) | [LockType Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Delete Method Example (VC++)

This example uses the [Delete](#) method to remove a specified record from a [Recordset](#).

```
// BeginDeleteCpp
#import "C:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <conio.h>
#include "DeleteX.h"

//Function Declarations.
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void DeleteX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);
inline char* mygets(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return NULL;

    if (!strrchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strrchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strrchr(strDest, '\n') - strDest] = '\0';

    return pstrRet;
}

////////////////////////////////////
//                                                                    //
//      Main Function                                                    //
//                                                                    //
////////////////////////////////////
```

```

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    DeleteX();

    // Wait here for the user to see the output
    printf("\n\nPress any key to continue...");
    getch();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//      DeleteX Function                                             //
//                                                                    //
////////////////////////////////////

void DeleteX(void)
{

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _RecordsetPtr  pRstRoySched = NULL;

    // Define Other Variables
    IADOResultBinding  *picRs = NULL; //Interface Pointer declared
    CRoySchedRs      royschrs;      //C++ class object
    HRESULT hr = S_OK;

    char  strTitleID[10], strTmpTitleID[10]="";
    long  longHiRange;
    int   intRoyalty, intLoRange, cnt=0;
    bool  blnFound = TRUE;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open RoySched table
        TESTHR(pRstRoySched.CreateInstance(__uuidof(Recordset)));

        pRstRoySched->CursorLocation = adUseClient;
        pRstRoySched->CursorType = adOpenStatic;
        pRstRoySched->LockType = adLockBatchOptimistic;
    }
}

```

```

TESTHR(pRstRoySched->Open("SELECT * FROM roysched WHERE"
    " royalty = 20",strCnn,adOpenStatic,adLockBatchOptimisti
    adCmdText));

// Prompt for a record to delete.
printf("Before delete there are %d titles with 20 percent "
    "royalty :\n", pRstRoySched->RecordCount);

// Open an IADORecordBinding interface pointer which we'll u
// for Binding Recordset to a class.
TESTHR(pRstRoySched->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

// Bind the Recordset to a C++ Class here
TESTHR(picRs->BindToRecordset(&royschrs));

while(!(pRstRoySched->EndOfFile))
{
    printf("%s\n", royschrs.lemp_titleidStatus == adFldOK ?
        royschrs.m_sz_titleid : "<NULL>");
    pRstRoySched->MoveNext();
}

printf("\nEnter the ID of a record to delete: ");
mygets(strTitleID, 10);

// Converting the title_id to upper case
for( cnt=0; cnt<10; cnt++)
{
    if(strTitleID[cnt] != NULL)
    {
        if( IsCharAlpha( strTitleID[cnt]) )
        {
            if( islower( strTitleID[cnt]) )
                strTmpTitleID[cnt] = _toupper(strTitleID[cnt]
            else
                strTmpTitleID[cnt] = strTitleID[cnt];
        }
        else
        {
            strTmpTitleID[cnt] = strTitleID[cnt];
        }
    }
}

// Move to the record and save data so it can be restored.
pRstRoySched->PutFilter ("title_id = '" +
    (_bstr_t)(LPCSTR)strTmpTitleID + "'");

```

```

if(pRstRoySched->RecordCount != 0)
{
    intLoRange = royschrs.m_sz_lorange;
    longHiRange = royschrs.m_sz_hirange;
    intRoyalty = royschrs.m_sz_royalty;

    // Delete the record
    pRstRoySched->Delete(adAffectCurrent);
    pRstRoySched->UpdateBatch(adAffectCurrent);
}
else
{
    blnFound = FALSE;
    printf("This Title ID not available");
}

// Show the results.
VARIANT varFilter;
varFilter.vt = VT_I2;
varFilter.iVal = adFilterNone;
pRstRoySched->PutFilter (varFilter);
pRstRoySched->Requery(-1);

// Bind the Recordset to a C++ Class here.
TESTHR(picRs->BindToRecordset(&royschrs));

printf("\nAfter delete there are %d titles with 20 percent"
       " royalty: ", pRstRoySched->RecordCount);

while(!(pRstRoySched->EndOfFile))
{
    printf("\n%s", royschrs.lemp_titleidStatus == adFldOK ?
          royschrs.m_sz_titleid : "<NULL>");
    pRstRoySched->MoveNext();
}

// Restore the data because this is a demonstration.
if(blnFound)
{
    // Set the final character of the destination string to
    royschrs.m_sz_titleid[sizeof(royschrs.m_sz_titleid)-1] =
    // The source string will get truncated if its length is
    // longer than the length of the destination string minu
    strncpy(royschrs.m_sz_titleid, strTmpTitleID, sizeof(roysc
    royschrs.m_sz_lorange = intLoRange;
    royschrs.m_sz_hirange = longHiRange;
    royschrs.m_sz_royalty = intRoyalty;

    TESTHR(picRs->AddNew(&royschrs));
}

```

```

        pRstRoySched->UpdateBatch(adAffectCurrent);
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstRoySched->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstRoySched)
    if (pRstRoySched->State == adStateOpen)
        pRstRoySched->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                             //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {

```

```

        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintComError Function                                         //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndDeleteCpp

```

DeleteX.h

```

// BeginDeleteH
#include "icrsint.h"

// This Class extracts titleid, lorange, hirange and royalty
// from the "roysched" table.

class CRoySchedRs : public CADOResultBinding
{
    BEGIN_ADO_BINDING(CRoySchedRs)

        //Column empid is the 1st field in the recordset

        ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_sz_titleid,
            sizeof(m_sz_titleid), lemp_titleidStatus, TRUE)

        ADO_VARIABLE_LENGTH_ENTRY2(2, adInteger, m_sz_lorange,

```

```
        sizeof(m_sz_lorange), lemp_lorangeStatus, TRUE)

ADO_VARIABLE_LENGTH_ENTRY2(3, adInteger, m_sz_hirange,
    sizeof(m_sz_hirange), lemp_hirangeStatus, TRUE)

ADO_VARIABLE_LENGTH_ENTRY2(4, adInteger, m_sz_royalty,
    sizeof(m_sz_royalty), lemp_royaltyStatus, TRUE)

END_ADO_BINDING()

public:

    CHAR        m_sz_titleid[10];
    ULONG       lemp_titleidStatus;
    ULONG       m_sz_lorange;
    ULONG       lemp_lorangeStatus;
    ULONG       m_sz_hirange;
    ULONG       lemp_hirangeStatus;
    ULONG       m_sz_royalty;
    ULONG       lemp_royaltyStatus;
};
// EndDeleteH
```

See Also

[Delete Method \(ADO Recordset\)](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState Properties Example (VC++)

This example triggers an error, traps it, and displays the [Description](#), [HelpContext](#), [HelpFile](#), [NativeError](#), [Number](#), [Source](#), and [SQLState](#) properties of the resulting [Error](#) object.

```
// BeginDescriptionCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void DescriptionX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    DescriptionX();

    //Wait here for user to see the output..
    printf("\nPress any key to continue...");
    getch();
    ::CoUninitialize();
}
```

```

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      DescriptionX Function                                                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void DescriptionX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _ConnectionPtr pConnection = NULL;
    ErrorPtr errorLoop = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;

    try
    {
        // Intentionally trigger an error.
        // open connection
        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));

        if (FAILED(hr = pConnection->Open("nothing","","",adConnectU
        {
            _com_issue_error(hr);
            exit(1);
        }

        // Cleanup object before exit.
        pConnection->Close();
    }
    catch(_com_error)
    {
        // Pass a connection pointer.
        PrintProviderError(pConnection);
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{

```

```

//Define Other Variables
HRESULT hr = S_OK;
_bstr_t strError;
ErrorPtr pErr = NULL;

try
{
    // Enumerate Errors collection and display
    // properties of each Error object.
    long nCount = pConnection->Errors->Count;

    // Collection ranges from 0 to nCount - 1.
    for(long i = 0; i < nCount; i++)
    {
        pErr = pConnection->Errors->GetItem(i);
        printf("Error #%d\n", pErr->Number);
        printf(" %s\n", (LPCSTR)pErr->Description);
        printf(" (Source: %s)\n", (LPCSTR)pErr->Source);
        printf(" (SQL State: %s)\n", (LPCSTR)pErr->SQLState);
        printf(" (NativeError: %d)\n", (LPCSTR)pErr->NativeError)
        if ((LPCSTR)pErr->GetHelpFile() == NULL)
        {
            printf("\tNo Help file available\n");
        }
        else
        {
            printf("\t(HelpFile: %s\n)", pErr->HelpFile);
            printf("\t(HelpContext: %s\n)", pErr->HelpContext);
        }
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    PrintComError(e);
}
}

////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());
}

```

```
// Print Com errors.

printf("Error\n");
printf("\tCode = %08lx\n", e.Error());
printf("\tCode meaning = %s", e.ErrorMessage());
printf("\tSource = %s\n", (LPCSTR) bstrSource);
printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndDescriptionCpp
```

See Also

[Description Property](#) | [Error Object](#) | [HelpContext Property](#) | [HelpFile Property](#) | [NativeError Property](#) | [Number Property](#) | [Source Property \(ADO Error\)](#) | [SQLState Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

{
HRESULT    hr = S_OK;

    // Define string variables.
    _bstr_t strSQLChange("UPDATE Titles SET Type = "
        "'self_help' WHERE Type = 'psychology'");
    _bstr_t strSQLRestore("UPDATE Titles SET Type = "
        "'psychology' WHERE Type = 'self_help'");
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _ConnectionPtr    pConnection = NULL;
    _CommandPtr       pCmdChange  = NULL;
    _RecordsetPtr     pRstTitles  = NULL;

    try
    {
        // Open connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        pConnection->Open (strCnn, "", "", adConnectUnspecified);

        // Create command object.
        TESTHR(pCmdChange.CreateInstance(__uuidof(Command));
        pCmdChange->ActiveConnection = pConnection;
        pCmdChange->CommandText = strSQLChange;

        // Open titles table, casting Connection pointer to an
        // IDispatch type so converted to correct type of variant.
        TESTHR(pRstTitles.CreateInstance(__uuidof(Recordset));
        pRstTitles->Open ("Titles", _variant_t((IDispatch *) pConnec
            true), adOpenStatic, adLockOptimistic, adCmdTable);

        // Print report of original data.
        printf(
            "\n\nData in Titles table before executing the query: \n

        // Call function to print loop recordset contents.
        PrintOutput(pRstTitles);

        // Clear extraneous errors from the Errors collection.
        pConnection->Errors->Clear();

        // Call ExecuteCommand subroutine to execute pCmdChange comm
        ExecuteCommand(pCmdChange, pRstTitles);

        // Print report of new data.
        printf(

```

```

        "\n\n\tData in Titles table after executing the query: \
PrintOutput(pRstTitles);

// Use the Connection object's execute method to
// execute SQL statement to restore data.
pConnection->Execute(strSQLRestore, NULL, adExecuteNoRecords

// Retrieve the current data by requerying the recordset.
pRstTitles->Requery(adCmdUnknown);

// Print report of restored data.
printf(
    "\n\n\tData after exec. query to restore original info:
PrintOutput(pRstTitles);
}
catch (_com_error &e)
{
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
if (pRstTitles)
    if (pRstTitles->State == adStateOpen)
        pRstTitles->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

////////////////////////////////////
//      ExecuteCommand Function      //
////////////////////////////////////

void ExecuteCommand(_CommandPtr pCmdTemp, _RecordsetPtr pRstTemp)
{
    try
    {
        // CommandText property already set before function was call
        pCmdTemp->Execute(NULL, NULL, adCmdText);

        // Retrieve the current data by requerying the recordset.
        pRstTemp->Requery(adCmdUnknown);
    }

    catch(_com_error &e)
    {
        // Notify user of any errors that result from
        // executing the query.

```

```

        // Pass a connection pointer accessed from the Recordset.
        PrintProviderError(pRstTemp->GetActiveConnection());
        PrintComError(e);
    }
}

////////////////////////////////////
//      PrintOutput Function      //
////////////////////////////////////

void PrintOutput(_RecordsetPtr pRstTemp)
{
    // Ensure at top of recordset.
    pRstTemp->MoveFirst();

    // If EOF is true, then no data and skip print loop.
    if( pRstTemp->EndOfFile )
    {
        printf("\tRecordset empty\n");
    }
    else
    {
        // Define temporary strings for output conversions.
        // Initialize to first record's values.
        _bstr_t bstrTitle;
        _bstr_t bstrType;

        // Enumerate Recordset and print from each.
        while(!(pRstTemp->EndOfFile))
        {
            // Convert variant string to convertible string type.
            bstrTitle = pRstTemp->Fields->GetItem("Title")->Value;
            bstrType = pRstTemp->Fields->GetItem("Type")->Value;
            printf("\t%s, %s \n",
                (LPCSTR) bstrTitle,
                (LPCSTR) bstrType);

            pRstTemp->MoveNext();
        }
    }
}

////////////////////////////////////
//      PrintProviderError Function      //
////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.

```

```

ErrorPtr  pErr = NULL;

if( (pConnection->Errors->Count) > 0)
{
    long nCount = pConnection->Errors->Count;
    // Collection ranges from 0 to nCount -1.
    for(long i = 0; i < nCount; i++)
    {
        pErr = pConnection->Errors->GetItem(i);
        printf("\t Error number: %x\t%s", pErr->Number,
            pErr->Description);
    }
}

////////////////////////////////////
//      PrintComError Function      //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndExecuteCpp

```

See Also

[Clear Method](#) | [Command Object](#) | [Connection Object](#) | [Errors Collection](#) | [Execute Method \(ADO Command\)](#) | [Execute Method \(ADO Connection\)](#) | [Requery Method](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Filter and RecordCount Properties Example (VC++)

This example uses the [Filter](#) property to open a new [Recordset](#) based on a specified condition applied to an existing **Recordset**. It uses the [RecordCount](#) property to show the number of records in the two **Recordsets**. The FilterField function is required for this procedure to run.

```
// BeginFilterCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <conio.h>
#include "FilterX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);}
void FilterX(void);
_RecordsetPtr FilterField(_RecordsetPtr rstTemp, _bstr_t strField, _
void FilterX2(void);
void PrintProviderError(_ConnectionPtr pCnn1);
void PrintComError(_com_error &e);
inline char* mygets(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return NULL;

    if (!strrchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strrchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strrchr(strDest, '\n') - strDest] = '\0';

    return pstrRet;
}
```

```

}

void main()
{
    HRESULT hr = S_OK;

    if(FAILED(::CoInitialize(NULL)))
        return;

    FilterX();

    //Wait here for user to see the output..
    printf("\nPress any key to Continue...");
    getch();

    //Clear the screen for the next display
    system("cls");

    FilterX2();

    //Wait here for user to see the output..
    printf("\nPress any key to Exit...");
    getch();
    ::CoUninitialize();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      FilterX Function                                                                           //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void FilterX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO_DB:: namespace.
    _RecordsetPtr rstPublishers = NULL;
    _RecordsetPtr rstPublishersCountry = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");
    int intPublisherCount = 0;
    long recCount = 0;
    _bstr_t strCountry;
    _bstr_t strMessage;
    char *tempStr;
    CHAR sz_CountryName[50];

```

```

bool boolFlag = TRUE;
try
{
    // Open recordset with data from Publishers table.
    rstPublishers.CreateInstance(__uuidof(Recordset));
    rstPublishersCountry.CreateInstance(__uuidof(Recordset));

    rstPublishers->CursorType = adOpenStatic;

    TESTHR(rstPublishers->Open("publishers",strCnn,
        adOpenStatic , adLockReadOnly,adCmdTable));

    // Populate the Recordset.
    intPublisherCount = rstPublishers->RecordCount;

    // Get user input.
    printf("Enter a country to filter on:");
    mygets(sz_CountryName, 50);

    // Trim the string
    tempStr = strtok(sz_CountryName, " \t");
    strCountry = tempStr;
    if (tempStr == NULL)
    {
        boolFlag = FALSE;
    }

    if (boolFlag)
    {
        if (strcmp(sz_CountryName,""))
        {
            // Open a filtered Recordset object.
            rstPublishersCountry = FilterField(rstPublishers,
                "Country", strCountry);
            recCount = rstPublishersCountry->GetRecordCount();
            if (recCount==0)
            {
                printf("\nNo publishers from that country.\n");
            }
            else
            {
                // Print number of records for the original
                // Recordset object and the filtered Recordset
                // object.
                printf("\nOrders in original recordset:\n%d",
                    intPublisherCount);
                printf("\nOrders in filtered recordset "
                    "(Country = '%s'): \n%d\n\n", (LPCSTR)strCou
                    rstPublishersCountry->RecordCount);
            }
        }
    }
}

```

```

        }
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect = rstPublishers->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
if (rstPublishers)
    if (rstPublishers->State == adStateOpen)
        rstPublishers->Close();
if (rstPublishersCountry)
    if (rstPublishersCountry->State == adStateOpen)
        rstPublishersCountry->Close();
}

_RecordsetPtr FilterField(_RecordsetPtr rstTemp, _bstr_t strField,
    _bstr_t strFilter)
{
    // Set a filter on the specified Recordset object and then
    // open a new Recordset object.
    rstTemp->Filter = strField + " = '" + strFilter + "'";
    return rstTemp;
}

void FilterX2(void)
{
    _RecordsetPtr rstPublishers;
    CPublishers publishers;

    //Define Other Variables
    HRESULT hr = S_OK;

```

```

IADORecordBinding *picRs = NULL; //Interface Pointer declared

_bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");
try
{
    // Open recordset with data from Publishers table.
    rstPublishers.CreateInstance(__uuidof(Recordset));

    rstPublishers->CursorType = adOpenStatic;

    TESTHR(rstPublishers->Open("SELECT * FROM publishers WHERE "
                              "Country='USA'", strCnn, adOpenStatic, adLockReadOnly,
                              adCmdText));

    //Open an IADORecordBinding interface pointer
    //which we'll use for Binding Recordset to a class
    TESTHR(rstPublishers->QueryInterface(
        __uuidof(IADORecordBinding), (LPVOID*)&picRs));

    //Bind the Recordset to a C++ Class here
    TESTHR(picRs->BindToRecordset(&publishers));

    // Print current data in recordset.
    rstPublishers->MoveFirst();

    while (!rstPublishers->EndOfFile)
    {
        printf("%s, %s\n",
            publishers.lP_pubnameStatus == adFldOK ?
            publishers.m_szP_pubname: "<NULL>",
            publishers.lP_countryStatus == adFldOK ?
            publishers.m_szP_country: "<NULL>");

        rstPublishers->MoveNext();
    }
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    _variant_t vtConnect = rstPublishers->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
    }
}

```



```

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndFilterCpp

```

FilterX.h

```

// BeginFilterH
#include "icrsint.h"

//This Class extracts only Pub Name and Country Name.
class CPublishers : public CADORecordBinding
{
BEGIN_ADO_BINDING(CPublishers)

    //Column Pub Name is the 2nd field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szP_pubname,
        sizeof(m_szP_pubname), lP_pubnameStatus, TRUE)

    //Column Country Name is the 5th field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(5, adVarChar, m_szP_country ,
        sizeof(m_szP_country), lP_countryStatus, TRUE)

END_ADO_BINDING()

public:
    CHAR    m_szP_pubname[50];
    ULONG   lP_pubnameStatus;
    CHAR    m_szP_country[50];
    ULONG   lP_countryStatus;
};
// EndFilterH

```

See Also

[Filter Property](#) | [RecordCount Property](#) | [Recordset Object](#)

ADO 2.5 Samples 

Find Method Example (VC++)

This example uses the [Recordset](#) object's [Find](#) method to locate and count the number of business titles in the **Pubs** database. The example assumes the underlying [provider](#) does not support similar functionality.

```
// BeginFindCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include "FindX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void FindX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    FindX();

    //Wait here for the user to see the output.
    printf("Press any key to continue...");
    getch();
    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//      FindX Function                                                //
//                                                                    //
////////////////////////////////////
```

```

void FindX(void)
{
    HRESULT    hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _ConnectionPtr    pConnection    = NULL;
    _RecordsetPtr     pRstTitles     = NULL;
    IADORecordBinding *picRs        = NULL; //Interface Pointer decl
    CTitlesRs titlers;              //C++ class object

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        pConnection->Open (strCnn, "", "", adConnectUnspecified);

        // Open title Table
        TESTHR(pRstTitles.CreateInstance(__uuidof(Recordset)));

        pRstTitles->Open("SELECT title_id FROM titles",
            _variant_t((IDispatch *)pConnection),
            adOpenStatic, adLockReadOnly, adCmdText);

        // The default parameters are sufficient to search forward
        // through a Recordset.

        pRstTitles->Find ("title_id LIKE 'BU%'",0,adSearchForward,""

        //Open an IADORecordBinding interface pointer which
        //we'll use for Binding Recordset to a class
        TESTHR(pRstTitles->QueryInterface(
            __uuidof(IADORecordBinding), (LPVOID*)&picRs));

        //Bind the Recordset to a C++ Class here
        TESTHR(picRs->BindToRecordset(&titlers));

        // Skip the current record to avoid finding the same
        // row repeatedly. The bookmark is redundant because Find
        // searches from the current position.
        int count = 0;

        //Continue if last find succeeded.
        while (!(pRstTitles->EndOfFile))
        {

```

```

        printf("Title ID: %s\n",titlers.lt_titleidStatus == adFl
            titlers.m_szt_titleid : "<NULL>");
        count++; //Count the last title fo

        _variant_t mark = pRstTitles->Bookmark; //Note current
        pRstTitles->Find("title_id LIKE 'BU'", 1, adSearchForwa
            mark);
    }
    printf("The number of business titles is %d\n",count);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstTitles)
    if (pRstTitles->State == adStateOpen)
        pRstTitles->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {

```

```

                pErr = pConnection->Errors->GetItem(i);
                printf("\t Error number: %x\t%s", pErr->Number,
                    (LPCSTR)pErr->Description);
            }
        }
    }

////////////////////////////////////
//                                                                    //
//          PrintComError Function                                     //
//                                                                    //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndFindCcpp

```

FindX.h

```

// BeginFindH
#include "icrsint.h"

//This Class extracts only titleId from Titles table.
class CTitlesRs : public CADOResultBinding
{
BEGIN_ADO_BINDING(CTitlesRs)

    // Column title_id is the 1st field in the recordset
    // from Titles table.
    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_szt_titleid,
        sizeof(m_szt_titleid), lt_titleidStatus, FALSE)

END_ADO_BINDING()

public:
    CHAR    m_szt_titleid[150];
    ULONG   lt_titleidStatus;
};
// EndFindH

```

See Also

[Find Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

GetRows Method Example (VC++)

This example uses the [GetRows](#) method to retrieve a specified number of rows from a [Recordset](#) and to fill an array with the resulting data. The **GetRows** method will return less than the desired number of rows in two cases: either if [EOF](#) has been reached, or if **GetRows** tried to retrieve a record that was deleted by another user. The function returns **False** only if the second case occurs. The `GetRowsOK` function is required for this procedure to run.

```
// BeginGetRowsCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <conio.h>

// Function Declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void GetRowsX(void);
bool GetRowsOK(_RecordsetPtr pRstTemp,int intNumber,
    _variant_t& avarData);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    GetRowsX();

    ::CoUninitialize();
}

//////////////////////////////////////
//                                     //
//      GetRowsX Function              //
//                                     //
//////////////////////////////////////

void GetRowsX(void)
{
    HRESULT hr = S_OK;
```

```

// Define string variables.
_bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
_RecordsetPtr pRstEmployees = NULL;

try
{
    // Open recordset with names and hire dates from employee ta
    TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));

    pRstEmployees->Open("SELECT fName, lName, hire_date "
                      "FROM Employee ORDER BY lName",strCnn,
                      adOpenStatic, adLockReadOnly,adCmdText);

    while (true) //continuous loop
    {
        int intLines = 0;

        // Get user input for number of rows.
        printf("\nEnter number of rows to retrieve (0 to exit):
        int intRows;
        scanf("%d", &intRows);

        if (intRows <= 0)
            break;

        //Clear the screen for the next display
        system("cls");

        // If GetRowsOK is successful, print the results,
        // noting if the end of the file was reached.
        _variant_t avarRecords;

        if (GetRowsOK(pRstEmployees, intRows, avarRecords))
        {
            long lUbound;
            HRESULT hr = SafeArrayGetUBound(avarRecords.parray,
            2,&lUbound);

            if (hr == 0)
            {
                if (intRows > lUbound + 1)
                {

```

```

        printf("\n(Not enough records in Recordset t
            \"retrieve %d rows)\n\", intRows);
    }
}
printf("%d records found.", lUbound+1);

// Print the retrieved data.
for (int intRecords = 0;intRecords < lUbound+1;
    intRecords++)
{
    printf("\n ");

    long rgIndices[2];
    rgIndices[0] = 0;
    rgIndices[1] = intRecords;
    _variant_t result;
    result.vt = VT_BSTR;

    hr= SafeArrayGetElement(avarRecords.parray,
        rgIndices, &result);

    if (hr == 0){printf("%s ",(LPCSTR)(_bstr_t)result

    rgIndices[0] = 1;

    hr= SafeArrayGetElement(avarRecords.parray,
        rgIndices, &result);

    if (hr == 0){printf("%s, ",(LPCSTR)(_bstr_t)result

    rgIndices[0] = 2;

    hr= SafeArrayGetElement(avarRecords.parray,
        rgIndices, &result);

    if (hr == 0){printf("%s", (LPCSTR)(_bstr_t)result

    intLines ++;

    if (intLines % 10 == 0)
    {
        printf("\nPress any key to continue...");
        getch();

        intLines = 0;

        //Clear the screen for the next display
        system("cls");
    }
}

```

```

    }
}
else
{
    // Assuming the GetRows error was due to data
    // changes by another user, use Requery to
    // refresh the Recordset and start over.
    printf("GetRows failed--retry?\n");
    char chKey;
    do
    {
        chKey = getch();
    }while(toupper(chKey) != 'Y'  && toupper(chKey) != '

    if(toupper(chKey) == 'Y')
    {
        pRstEmployees->Requery(adOptionUnspecified);
    }
    else
    {
        printf("GetRows failed!\n");
        break;
    }
}

    // Because using GetRows leaves the current
    // record pointer at the last record accessed,
    // move the pointer back to the beginning of the
    // Recordset before looping back for another search.
    pRstEmployees->MoveFirst();
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstEmployees->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
    }
}

```

```

        break;
    }
}

//Clean up objects before exit.
if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
}

bool GetRowsOK(_RecordsetPtr pRstTemp,int intNumber,
    _variant_t& avarData)
{
    // Store results of GetRows method in array.
    avarData = pRstTemp->GetRows(intNumber);

    // Return False only if fewer than the desired
    // number of rows were returned, but not because the
    // end of the Recordset was reached.
    long lUbound;
    HRESULT hr = SafeArrayGetUBound(avarData.parray,2,&lUbound);
    if (hr == 0)
    {
        if ((intNumber > lUbound + 1) && (!(pRstTemp->EndOfFile)))
            return false;
        else
            return true;
    }
    else
    {
        printf ("\nUnable to Get the Array's Upper Bound\n");
        return false;
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {

```

```

        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndGetRowsCpp

```

See Also

[BOF, EOF Properties](#) | [GetRows Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _ConnectionPtr pConnection = NULL;
    _RecordsetPtr pRstAuthors = NULL;

    // Define string variables.
    _bstr_t strCnn("Provider='sqloledb';Data Source='localhost';"
        "Initial Catalog='Northwind';Integrated Security='SSPI'");
    _bstr_t varOutput;
    char *strPrompt = "Enter a state (CA, IN, KS, MD, MI, OR, TN, UT
    char strState[2], *pState;

    try
    {
        // Open connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));

        // Open a command object for a stored procedure,
        // with one parameter.
        TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));

        printf("%s", strPrompt);
        gets(strState);

        pState = strtok(strState, " \t");

        char strQry[100] = "SELECT au_fname, au_lname, address, city
            "FROM authors WHERE state = ";

        strcat(strQry, "");
        strcat(strQry, pState);
        strcat(strQry, "");

        _bstr_t strQuery(strQry);

        pConnection->Open (strCnn, "", "", adConnectUnspecified);

        pRstAuthors->Open(strQuery, _variant_t((IDispatch*)pConnecti
            true),
            adOpenStatic, adLockReadOnly, adCmdText);

        if (pRstAuthors->RecordCount > 0)
        {
            // Use all defaults: get all rows, TAB column delimiter,
            // CARRIAGE RETURN row delimiter, empty-string null deli
            long lRecCount = pRstAuthors->RecordCount;

```

```

        _bstr_t varTab("\t");
        _bstr_t varRet("\r");
        _bstr_t varNull("");
        varOutput = pRstAuthors->GetString(adClipString,lRecCount,
            varTab,varRet,varNull);
        printf("State = '%s'\n", strState);
        printf ("Name          Address          City\n");
        printf ("%s\n", (LPCTSTR)varOutput);
    }
    else
    {
        printf("\nNo rows found for state = '%s'\n",pState);
    }
    // Clean up objects before exit.
    pRstAuthors->Close();
    pConnection->Close();
}

catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    PrintProviderError(pConnection);
    PrintComError(e);
}
}

```

```

////////////////////////////////////
//                                                                    //
//          PrintProviderError Function                               //
//                                                                    //
////////////////////////////////////

```

```

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

```

```

    }
}

////////////////////////////////////
//                                                                    //
//      PrintComError Function                                       //
//                                                                    //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndGetStringCpp

```

See Also

[GetString Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

//                                                    //
//      IsolationLevelX()  Function                    //
//                                                    //
//                                                    //
////////////////////////////////////////////////////////////////////

void IsolationLevelX(void)
{
    // Define ADO ObjectPointers
    // Initialize Pointers on define
    // These are in the ADODB :: namespace
    _RecordsetPtr  pRstTitles  = NULL;
    _ConnectionPtr pConnection = NULL;

    // Define other Variables
    HRESULT hr = S_OK;
    IADORecordBinding *picRs  = NULL; // Interface Pointer Declared
    CTitleRs titlers;           // C++ Class Object
    LPSTR p_TempStr = NULL;

    //Assign Connection String to Variable
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
                  "Initial Catalog='pubs';Integrated Security='SSP

try
{
    // Open Connection and Titles Table
    TESTHR(pConnection.CreateInstance(__uuidof(Connection));
    pConnection->Mode = adModeShareExclusive;
    pConnection->IsolationLevel = adXactIsolated;
    pConnection->Open(strCnn, "", "", adConnectUnspecified);

    TESTHR(pRstTitles.CreateInstance(__uuidof(Recordset)));
    pRstTitles->CursorType = adOpenDynamic;
    pRstTitles->LockType = adLockPessimistic;

    pRstTitles->Open("titles",_variant_t((IDispatch*) pConnectio
        true),adOpenDynamic,adLockPessimistic,adCmdTable);

    pConnection->BeginTrans();

    // Display Connection Mode
    if(pConnection->Mode == adModeShareExclusive)
    {
        printf("Connection Mode Is Exclusive");
    }
    else
    {
        printf("Connection Mode Is Not Exclusive");
    }
}

```

```

// Display Isolation Level
if(pConnection->IsolationLevel == adXactIsolated)
{
    printf("\n\nTransaction is Isolated");
    printf("\n\nPress any key to continue...\n\n");
    getch();
}
else
{
    printf("\n\nTransaction is not Isolated");
    printf("\n\nPress any key to continue...\n\n");
    getch();
}

// Open an IADORecordBinding interface pointer which
// we will use for binding Recordset to a class
TESTHR(pRstTitles->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

// Bind the Recordset to a C++ class here
TESTHR(picRs->BindToRecordset(&titlers));

// Change the type of psychology titles.
p_TempStr = (LPSTR) malloc(sizeof(titlers.m_szau_Type));

while (!(pRstTitles->EndOfFile))
{
    // Set the final character of the destination string to
    p_TempStr[sizeof(titlers.m_szau_Type)-1] = '\0';
    // The source string will get truncated if its length is
    // longer than the length of the destination string minu
    strncpy(p_TempStr, strtok(titlers.m_szau_Type, " "), sizeof

    // Compare type with psychology
    if (!strcmp(p_TempStr, "psychology"))
    {
        // Set the final character of the destination string
        titlers.m_szau_Type[sizeof(titlers.m_szau_Type)-1] =
        // Copy "self_help" title field
        // The source string will get truncated if its lengt
        // longer than the length of the destination string
        strncpy(titlers.m_szau_Type, "self_help", sizeof(title
        picRs->Update(&titlers);
    }
    pRstTitles->MoveNext();
}
// Print current data in recordset.
pRstTitles->Requery(adOptionUnspecified);

```

```

// Open again IADORecordBinding interface pointer for Bindin
// Recordset to a class.
TESTHR(pRstTitles->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

// ReBinding the Recordset to a C++ Class
TESTHR(picRs->BindToRecordset(&titlers));

// Move to the first record of the title table
pRstTitles->MoveFirst();

//Clear the screen for the next display
system("cls");

while (!pRstTitles->EndOfFile)
{
    printf("%s - %s\n", titlers.lau_TitleStatus == adFldOK ?
        titlers.m_szau_Title : "<NULL>",
        titlers.lau_TypeStatus == adFldOK ?
        titlers.m_szau_Type : "<NULL>");
    pRstTitles->MoveNext();
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstTitles)
    if (pRstTitles->State == adStateOpen)
        pRstTitles->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
    {
        // Restore Original Data
        pConnection->RollbackTrans();

        pConnection->Close();
    }
}

// Deallocate the memory
if (p_TempStr)
    free(p_TempStr);

```

```

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//          PrintProviderError Function                                                           //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    //Print Provider Errors from Connection object
    //pErr is a record object in the Connection's Error collection
    ErrorPtr    pErr = NULL;

    if((pConnection->Errors->Count)>0)
    {
        long nCount = pConnection->Errors->Count;

        //Collection ranges from 0 to nCount-1
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error Number :%x \t %s",pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//          PrintComError Function                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndIsolationLevelCpp

```

IsolationLevelX.h

```

// BeginIsolationLevelH

#include "icrsint.h"

//This class extracts titles and type from Title table

class CTitleRs : public CADOResultBinding
{
BEGIN_ADO_BINDING(CTitleRs)
    // Column title is the 2nd field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(2,adVarChar,m_szau_Title,
        sizeof(m_szau_Title),lau_TitleStatus,FALSE)
    // Column type is the 3rd field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(3,adVarChar,m_szau_Type,
        sizeof(m_szau_Type),lau_TypeStatus,TRUE)
END_ADO_BINDING()

public:
    CHAR m_szau_Title[81];
    ULONG lau_TitleStatus;
    CHAR m_szau_Type[13];
    ULONG lau_TypeStatus;
};
// EndIsolationLevelH

```

See Also

[IsolationLevel Property](#) | [Mode Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Item Property Example (VC++)

This example demonstrates how the [Item](#) property accesses members of a collection. The example opens the *Authors* table of the *Pubs* database with a parameterized command.

The parameter in the command issued against the database is accessed from the [Command](#) object's [Parameters](#) collection by index and name. Then the fields of the returned [Recordset](#) are accessed from that object's [Fields](#) collection by index and name.

```
// BeginItemCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void ItemX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                               //
//      Main Function             //
//                               //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    ItemX();

    //Wait here for the user to see the output
    printf("Press any key to continue...");
    getch();
    ::CoUninitialize();
}
```



```

pCmd->Parameters->Append(pPrm);

pConnection->Open(strCnn, "", "", adConnectUnspecified);
pCmd->ActiveConnection = pConnection;

// Connection and CommandType are omitted because
// a Command object is provided.
_variant_t Conn;
Conn.vt = VT_ERROR;
Conn.scode = DISP_E_PARAMNOTFOUND;

pRst->Open((_variant_t((IDispatch *) pCmd)),Conn,adOpenStati
          adLockReadOnly, -1);

printf("The Parameters collection accessed by index...\n");

vIndex = (short) 0;
pPrm = pCmd->Parameters->GetItem(&vIndex);
printf("Parameter name = '%s', value = '%s'\n",
       (LPCSTR)pPrm->Name, (LPSTR)(_bstr_t)pPrm->Value);

printf("\nThe Parameters collection accessed by name...\n");
pPrm = pCmd->Parameters->Item["ItemXparm"];
printf("Parameter name = '%s', value = '%s'\n",
       (LPCSTR)pPrm->Name, (LPSTR)(_bstr_t)pPrm->Value);

printf("\nThe Fields collection accessed by index...\n");
pRst->MoveFirst();
long limit = 0;
limit = ((pRst->Fields->Count) - 1);
int intLineCnt = 8;
for (short iIndex = 0; iIndex <= limit; iIndex++)
{
    vIndex = iIndex;
    intLineCnt++;
    if (intLineCnt%15 == 0)
    {
        printf("\nPress any key to continue...\n");
        getch();
    }
    pFld = pRst->Fields->GetItem(&vIndex);
    printf("Field %d : Name = '%s', ", iIndex,
          (LPCSTR)pFld->GetName());
    _variant_t FldVal = pFld->GetValue();

    // Because Value is the default property of a
    // Property object,the use of the actual keyword
    // here is optional.
    switch(FldVal.vt)

```

```

    {
        case (VT_BOOL):
            if(FldVal.boolVal)
            {
                printf("Value = 'True'");
            }
            else
            {
                printf("Value = 'False'");
            }
            printf("\n");
            break;
        case (VT_BSTR):
            printf("Value = '%s'",
                (LPCSTR)(_bstr_t)FldVal.bstrVal);
            printf("\n");
            break;
        case (VT_I4):
            printf("Value = '%s'", (LPCSTR)FldVal.iVal);
            printf("\n");
            break;
        case (VT_EMPTY):
            printf("Value = '%s'", (LPCSTR)FldVal.lVal);
            printf("\n");
            break;
        default:
            break;
    }
}

printf("\nThe Fields collection accessed by name...\n");
pRst->MoveFirst();
for (iIndex = 0; iIndex <= 8; iIndex++)
{
    intLineCnt++;
    if (intLineCnt%15 == 0)
    {
        printf("\nPress any key to continue...\n");
        getch();
    }
    pFld = pRst->Fields->GetItem(Column[iIndex]);
    printf("Field name = '%s', ", (LPCSTR)pFld->GetName());
    _variant_t FldVal = pFld->GetValue();

    // Because Value is the default property of a
    // Property object, the use of the actual keyword
    // here is optional.
    switch(FldVal.vt)
    {
        case (VT_BOOL):

```



```

////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////
//                                                                    //
//          PrintComError Function                                     //
//                                                                    //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndItemCpp

```

See Also

[Command Object](#) | [Fields Collection](#) | [Item Property](#) | [Parameters Collection](#) | [Recordset Object](#)

ADO 2.5 Samples 

MarshalOptions Property Example (VC++)

This example uses the [MarshalOptions](#) property to specify what rows are sent back to the server—All Rows or only Modified Rows.

```
// BeginMarshalOptionsCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <conio.h>
#include <malloc.h>
#include "MarshalOptionsX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void MarshalOptionsX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

//////////////////////////////////////
//                                     //
//             Main Function           //
//                                     //
//////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    MarshalOptionsX();

    printf("Press any key to continue...");
    getch();
    ::CoUninitialize();
}

//////////////////////////////////////
//                                     //
//             Marshal Options Function //
//                                     //
//////////////////////////////////////
```

```

////////////////////////////////////
void MarshalOptionsX(void)
{
    // Define string variables
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _RecordsetPtr pRstEmployees = NULL;

    // Define Other Variables
    IADORecordBinding *picRs = NULL; //Interface Pointer declare
    CEmployeeRs emprs; //C++ Class Object
    HRESULT hr = S_OK;
    LPSTR strOldFirst = NULL;
    LPSTR strOldLast = NULL;

    try
    {
        // Open recordset with names from Employee table.
        TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));
        pRstEmployees->CursorType = adOpenKeyset;
        pRstEmployees->LockType = adLockOptimistic;
        pRstEmployees->CursorLocation = adUseClient;
        pRstEmployees->Open("SELECT fname, lname FROM Employee "
            "ORDER BY lname",strCnn,
            adOpenKeyset, adLockOptimistic,adCmdText);

        // Open an IADORecordBinding interface pointer which
        // we'll use for Binding Recordset to a class.
        TESTHR(pRstEmployees->QueryInterface(
            __uuidof(IADORecordBinding),(LPVOID*)&picRs));

        // Bind the Recordset to a C++ Class here.
        TESTHR(picRs->BindToRecordset(&emprs));

        // Store Original Data
        strOldFirst = (LPSTR) malloc(sizeof(emprs.m_szemp_fname));
        strOldLast = (LPSTR) malloc(sizeof(emprs.m_szemp_lname));
        // Set the final character of the destination string to NULL
        strOldFirst[sizeof(emprs.m_szemp_fname)-1] = '\0';
        // The source string will get truncated if its length is
        // longer than the length of the destination string minus on
        strncpy(strOldFirst, strtok(emprs.m_szemp_fname, " "),
            sizeof(emprs.m_szemp_fname)-1);
        // Set the final character of the destination string to NULL
    }
}

```

```

strOldLast[sizeof(emprs.m_szemp_lname)-1] = '\\0';
// The source string will get truncated if its length is
// longer than the length of the destination string minus on
strncpy(strOldLast, strtok(emprs.m_szemp_lname, " "),
        sizeof(emprs.m_szemp_lname)-1);

//Change Data in Edit Buffer
// Set the final character of the destination string to NULL
emprs.m_szemp_fname[sizeof(emprs.m_szemp_fname)-1] = '\\0';
// The source string will get truncated if its length is
// longer than the length of the destination string minus on
strncpy(emprs.m_szemp_fname, "Linda", sizeof(emprs.m_szemp_fna
// Set the final character of the destination string to NULL
emprs.m_szemp_lname[sizeof(emprs.m_szemp_lname)-1] = '\\0';
// The source string will get truncated if its length is
// longer than the length of the destination string minus on
strncpy(emprs.m_szemp_lname, "Kobara", sizeof(emprs.m_szemp_lname

// Show contents of buffer and get user input
printf("Edit in Progress:\\n");
printf("Original Data = %s %s \\n", strOldFirst, strOldLast);
printf("Data in buffer = %s %s \\n", emprs.lemp_fnameStatus
        adFldOK ? emprs.m_szemp_fname : "<NULL>",
        emprs.lemp_lnameStatus == adFldOK ?
        emprs.m_szemp_lname : "<NULL>");
printf("Use Update to replace the original data with the "
        "buffered data in the Recordset?\\nEnter (y/n) :?");
char opt1=getch();

if(toupper(opt1)=='Y')
{
    printf("\\nWould you like to send all the rows in "
            "the recordset back to the server?\\nEnter (y/n):");
    char opt2 = getch();
    if(toupper(opt2) == 'Y')
    {
        {
            pRstEmployees->MarshalOptions = adMarshalAll
            picRs->Update(&emprs);
        }
    }
    else
    {
        printf("\\nWould you like to send only modified "
                "rows back to the server?\\nEnter (y/n):");
        char opt3=getch();
        if(toupper(opt3) == 'Y')
        {
            pRstEmployees->MarshalOptions =

```

```

                adMarshalModifiedOnly;
                picRs->Update(&emprs);
            }
        }
    }
    // Show the resulting data
    printf("\nData In the Recordset = %s %s\n",
        emprs.lemp_fnameStatus == adFldOK ?
        emprs.m_szemp_fname : "<NULL>",
        emprs.lemp_lnameStatus == adFldOK ?
        emprs.m_szemp_lname : "<NULL>");

    // Restore original data because this is a demonstration
    if((strcmp(strOldFirst,emprs.m_szemp_fname)) &&
        (strcmp(strOldLast,emprs.m_szemp_lname)))
    {
        // Set the final character of the destination string to
        emprs.m_szemp_fname[sizeof(emprs.m_szemp_fname)-1] = '\0
        // The source string will get truncated if its length is
        // longer than the length of the destination string minu
        strncpy(emprs.m_szemp_fname,strOldFirst,
            sizeof(emprs.m_szemp_fname)-1);
        // Set the final character of the destination string to
        emprs.m_szemp_lname[sizeof(emprs.m_szemp_lname)-1] = '\0
        // The source string will get truncated if its length is
        // longer than the length of the destination string minu
        strncpy(emprs.m_szemp_lname,strOldLast,
            sizeof(emprs.m_szemp_lname)-1);
        picRs->Update(&emprs);
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstEmployees->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occured.");
            break;
    }
}

```



```

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndMarshalOptionsCpp

```

MarshalOptionsX.h

```

// BeginMarshalOptionsH

#include "icrsint.h"

//This Class extracts only fname,lname from employees table
class CEmployeeRs : public CADOResultBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

    //Column fname is the 1st field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_szemp_fname,
        sizeof(m_szemp_fname), lemp_fnameStatus, TRUE)

    //Column lname is the 2nd field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szemp_lname,
        sizeof(m_szemp_lname), lemp_lnameStatus, TRUE)

END_ADO_BINDING()

public:
    CHAR    m_szemp_fname[21];
    ULONG    lemp_fnameStatus;
    CHAR    m_szemp_lname[31];
    ULONG    lemp_lnameStatus;
};
// EndMarshalOptionsH

```

See Also

[MarshalOptions Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

MaxRecords Property Example (VC++)

This example uses the [MaxRecords](#) property to open a [Recordset](#) containing the 10 most expensive titles in the *Titles* table.

```
// BeginMaxRecordsCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll"
    no_namespace rename("EOF","EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include "MaxRecordsX.h"

// Function Declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void MaxRecordsX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//                               Main Function                          //
//                                                                    //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    MaxRecordsX();

    printf("Press any key to continue...");
    getch();
    ::CoUninitialize();
}

// MaxRecordsX() Function
void MaxRecordsX(void)
{
    // Define ADO ObjectPointers
    // Initialize Pointers on define
```

```

// These are in the ADODB :: namespace
_RecordsetPtr pRstTemp    = NULL;

// Define Other Variables
IADORecordBinding *picRs  = NULL;    // Interface Pointer Declare
CTitleRs titlers;          // C++ Class Object
HRESULT hr = S_OK;

try
{
    //Assign Connection String to Variable
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer'
        'Initial Catalog='pubs';Integrated Security='SSP

// Open Recordset containing the 10 most expensive titles in
// Titles table.
TESTHR(pRstTemp.CreateInstance(__uuidof(Recordset)));

pRstTemp->MaxRecords=10;

pRstTemp->Open("SELECT title,price FROM Titles "
    "ORDER BY Price DESC",strCnn,adOpenForwardOnly,
    adLockReadOnly,adCmdText);

// Open an IADORecordBinding interface pointer which
// we will use for binding Recordset to a class
TESTHR(pRstTemp->QueryInterface(
    __uuidof(IADORecordBinding),(LPVOID*)&picRs));

// Bind the Recordset to a C++ class here
TESTHR(picRs->BindToRecordset(&titlers));

// Display the contents of the Recordset
printf("Top Ten Titles by Price:\n\n");

while(!(pRstTemp->EndOfFile))
{
    printf("%s --- %6.2lf\n\n",titlers.lau_TitleStatus ==
        adFldOK ? titlers.m_szau_Title : "<NULL>",
        titlers.lau_PriceStatus == adFldOK ?
        titlers.m_szau_Price : 0.00);
    pRstTemp->MoveNext();
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstTemp->GetActiveConnection();

```

```

// GetActiveConnection returns connect string if connection
// is not open, else returns Connection object.
switch(vtConnect.vt)
{
    case VT_BSTR:
        PrintComError(e);
        break;
    case VT_DISPATCH:
        PrintProviderError(vtConnect);
        break;
    default:
        printf("Errors occured.");
        break;
}
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstTemp)
    if (pRstTemp->State == adStateOpen)
        pRstTemp->Close();
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//          PrintProviderError Function                                                           //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    //Print Provider Errors from Connection object
    //pErr is a record object in the Connection's Error collection
    ErrorPtr    pErr = NULL;

    if((pConnection->Errors->Count)>0)
    {
        long nCount = pConnection->Errors->Count;

        //Collection ranges from 0 to nCount-1
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error Number :%x \t %s",pErr->Number,
                pErr->Description);
        }
    }
}

```

```

    }
}

////////////////////////////////////
//                                     //
//      PrintComError Function         //
//                                     //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndMaxRecordsCpp

```

See Also

[MaxRecords Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    MoveX();

    ::CoUninitialize();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      MoveX Function                                                //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void MoveX(void)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr pRstAuthors = NULL;

    // Define Other Variables
    IADORecordBinding *picRs = NULL; //Interface Pointer declared
    CAuthorsRs authorsrs; //C++ class object
    HRESULT hr = S_OK;

    // Open Authors table
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open recordset from Authors table.
        TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));

        pRstAuthors->CursorType = adOpenStatic;
        // Use client cursor to allow use of AbsolutePosition proper
        pRstAuthors->CursorLocation = adUseClient;

        pRstAuthors->Open("SELECT au_id, au_fname, au_lname, city, "
            "state FROM Authors ORDER BY au_lname", strCnn, adOpenSt
            adLockReadOnly, adCmdText);

        // Open an IADORecordBinding interface pointer which we'll u
        // for Binding Recordset to a class.
        TESTHR(pRstAuthors->QueryInterface(
            __uuidof(IADORecordBinding), (LPVOID*)&picRs));
    }
}

```

```

// Bind the Recordset to a C++ Class here.
TESTHR(picRs->BindToRecordset(&authorsrs));

pRstAuthors->MoveFirst();

char *strMove;
char strTemp[5];
while(true)
{
    // Display information about current record and ask how
    // records to move.
    printf("Record %ld of %d\n", pRstAuthors->AbsolutePositi
        pRstAuthors->RecordCount);
    printf("Author: %s %s\n",
        authorsrs.lemp_fnameStatus == adFldOK ?
        authorsrs.m_au_fname : "<NULL>",
        authorsrs.lemp_lnameStatus == adFldOK ?
        authorsrs.m_au_lname : "<NULL>");
    printf("Location: %s, %s\n\n",
        authorsrs.lemp_cityStatus == adFldOK ?
        authorsrs.m_au_city : "<NULL>",
        authorsrs.lemp_stateStatus == adFldOK ?
        authorsrs.m_au_state : "<NULL>");

    printf("Enter number of records to Move "
        "\n(positive or negative, Enter to quit): ");
    mygets(strTemp, 5);

    strMove = strtok(strTemp, " \t");

    if (strMove == NULL)
        break;

    // if the input is not numeric then notify the user.
    if(!atol(strMove))
    {
        printf("Expecting numeric value...\n");
        continue;
    }

    // Store bookmark in case the Move goes too far
    // forward or backward.
    _variant_t varBookmark = pRstAuthors->Bookmark;

    // Move method requires parameter of data type Long.
    long lngMove = atol(strMove);

    pRstAuthors->Move(lngMove);
}

```

```

        // Trap for BOF or EOF.
        if(pRstAuthors->BOF)
        {
            printf("Too far backward! Returning to current"
                " record.\n");
            pRstAuthors->Bookmark = varBookmark;
        }

        if(pRstAuthors->EndOfFile)
        {
            printf("Too far forward! Returning to current"
                " record.\n");
            pRstAuthors->Bookmark = varBookmark;
        }
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstAuthors->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
//Release the IAD0Recordset Interface here
if (picRs)
    picRs->Release();

if (pRstAuthors)
    if (pRstAuthors->State == adStateOpen)
        pRstAuthors->Close();
}

```

////////////////////////////////////

```

//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr  pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndMoveCpp

```

MoveX.h

```

// BeginMoveH

#include "icrsint.h"

```

```

// This Class extracts fname, lname, city and state
// from the "authors" table.

class CAuthorsRs : public CADOResultBinding
{
BEGIN_ADO_BINDING(CAuthorsRs)

    // Column au_id is the 1st field in the recordset

    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_au_fname,
        sizeof(m_au_fname), lemp_fnameStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_au_lname,
        sizeof(m_au_lname), lemp_lnameStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_au_city,
        sizeof(m_au_city), lemp_cityStatus, TRUE)

    ADO_VARIABLE_LENGTH_ENTRY2(5, adChar, m_au_state,
        sizeof(m_au_state), lemp_stateStatus, TRUE)

END_ADO_BINDING()

public:
    char    m_au_fname[21];
    ULONG   lemp_fnameStatus;
    char    m_au_lname[41];
    ULONG   lemp_lnameStatus;
    char    m_au_city[21];
    ULONG   lemp_cityStatus;
    char    m_au_state[3];
    ULONG   lemp_stateStatus;
};

// EndMoveH

```

See Also

[Move Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

HRESULT hr = S_OK;
_RecordsetPtr pRstAuthors = NULL;
_bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
    "Initial Catalog='pubs';Integrated Security='SSPI'");
_bstr_t strMessage("UPDATE Titles SET Type = "
    "'psychology' WHERE Type = 'self_help'");
int intCommand = 0;

// Temporary string variable for type conversion for printing.
_bstr_t bstrFName;
_bstr_t bstrLName;

try
{
    // Open recordset from Authors table.
    TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));
    pRstAuthors->CursorType = adOpenStatic;

    // Use client cursor to enable AbsolutePosition property.
    pRstAuthors->CursorLocation = adUseClient;
    pRstAuthors->Open("Authors", strCnn, adOpenStatic,
        adLockBatchOptimistic, adCmdTable);

    // Show current record information and get user's method cho
    while (true) // Continuous loop.
    {
        // Convert variant string to convertible string type.
        bstrFName = pRstAuthors->Fields->Item["au_fName"]->Value
        bstrLName = pRstAuthors->Fields->Item["au_lName"]->Value

        printf("Name: %s %s\n Record %d of %d\n\n",
            (LPCSTR) bstrFName,
            (LPCSTR) bstrLName,
            pRstAuthors->AbsolutePosition,
            pRstAuthors->RecordCount);
        printf("[1 - MoveFirst, 2 - MoveLast, \n");
        printf(" 3 - MoveNext, 4 - MovePrevious, 5 - Quit]\n");

        scanf("%d", &intCommand);

        if ((intCommand < 1) || (intCommand > 4))
            break; // Out of range entry exits program loop.

        // Call method based on user's input.
        MoveAny(intCommand, pRstAuthors);
    }
}
catch (_com_error &e)
{
    // Notify the user of errors if any.

```

```

// Pass a connection pointer accessed from the Recordset.
_variant_t vtConnect = pRstAuthors->GetActiveConnection();

// GetActiveConnection returns connect string if connection
// is not open, else returns Connection object.
switch(vtConnect.vt)
{
    case VT_BSTR:
        PrintComError(e);
        break;
    case VT_DISPATCH:
        PrintProviderError(vtConnect);
        break;
    default:
        printf("Errors occurred.");
        break;
}
}

// Clean up objects before exit.
if (pRstAuthors)
    if (pRstAuthors->State == adStateOpen)
        pRstAuthors->Close();
}

////////////////////////////////////
//      MoveAny Function      //
////////////////////////////////////

void MoveAny(int intChoice, _RecordsetPtr pRstTemp)
{
    // Use specified method, trapping for BOF and EOF
    try
    {
        switch(intChoice)
        {
            case 1:
                pRstTemp->MoveFirst();
                break;
            case 2:
                pRstTemp->MoveLast();
                break;
            case 3:
                pRstTemp->MoveNext();
                if (pRstTemp->EndOfFile)
                {
                    printf("\nAlready at end of recordset!\n");
                    pRstTemp->MoveLast();
                }
                //End If
        }
    }
}

```

```

        break;
    case 4:
        pRstTemp->MovePrevious();
        if (pRstTemp->BOF)
        {
            printf("\nAlready at beginning of recordset!\n")
            pRstTemp->MoveFirst();
        }
        break;
    default:
        ;
    }
}

catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstTemp->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

}

////////////////////////////////////
//      PrintProviderError Function      //
////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.

    // pErr is a record object in the Connection's Error collection.
    ErrorPtr pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {

```

```

        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount - 1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////
//      PrintComError Function      //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndMoveFirstCpp

```

See Also

[MoveFirst, MoveLast, MoveNext, and MovePrevious Methods](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

void NextRecordsetX(void)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr    pRstCompound = NULL;

    // Define Other Variables
    HRESULT hr = S_OK;
    _variant_t index;
    index.vt = VT_I2;

    // Assign connection string to a variable.
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open recordset from Authors table.
        TESTHR(pRstCompound.CreateInstance(__uuidof(Recordset)));

        // Pass the Cursor type and Lock type to the Recordset.
        pRstCompound->Open("SELECT * FROM authors; SELECT * FROM sto
            \"SELECT * FROM jobs\", strCnn, adOpenForwardOnly,
            adLockReadOnly, adCmdText);

        // Display results from each SELECT statement.
        int intCount = 1;
        while(!(pRstCompound==NULL))
        {
            printf("\n\nContents of recordset #d\n", intCount);

            while(!pRstCompound->EndOfFile)
            {
                index.iVal = 0;
                printf("%s\t", (LPCSTR)(_bstr_t)pRstCompound->\
                    GetFields()->GetItem(& index)->Value);
                index.iVal = 1;
                printf("%s\n", (LPCSTR)(_bstr_t)pRstCompound->\
                    Fields->GetItem(& index)->Value);

                pRstCompound->MoveNext();

                int intLine = intLine + 1;
                if (intLine % 22 == 0)
                {
                    printf("\nPress any key to continue...");
                    getch();

                    //Clear the screen for the next display.

```

```

        system("cls");
    }
}
long lngRec = 0;
pRstCompound = pRstCompound->
    NextRecordset((VARIANT *)lngRec);

printf("\nPress any key to continue...");
getch();
intCount = intCount + 1;
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstCompound->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
if (pRstCompound)
    if (pRstCompound->State == adStateOpen)
        pRstCompound->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.

```

```

// pErr is a record object in the Connection's Error collection.
ErrorPtr    pErr    = NULL;

if( (pConnection->Errors->Count) > 0)
{
    long nCount = pConnection->Errors->Count;
    // Collection ranges from 0 to nCount -1.
    for(long i = 0; i < nCount; i++)
    {
        pErr = pConnection->Errors->GetItem(i);
        printf("\t Error number: %x\t%s", pErr->Number,
            (LPCSTR) pErr->Description);
    }
}

////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndNextRecordsetCpp

```

See Also

[NextRecordset Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

void NumericScaleX(void)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace
    _RecordsetPtr pRstDiscounts = NULL;
    FieldsPtr fldTemp = NULL;

    //Define Other Variables
    HRESULT hr = S_OK;
    _variant_t Index;
    Index.vt = VT_I2;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open recordset.
        TESTHR(pRstDiscounts.CreateInstance(__uuidof(Recordset)));
        pRstDiscounts->Open("discounts", strCnn, adOpenForwardOnly,
            adLockReadOnly, adCmdTable);

        // Display numeric scale and precision of
        // numeric and small integer fields.
        fldTemp = pRstDiscounts->GetFields();

        for (int intLoop=0;intLoop < (int)fldTemp->GetCount();intLoop
        {
            Index.iVal = intLoop;

            if ((fldTemp->GetItem(Index)->Type == adNumeric)
                || (fldTemp->GetItem(Index)->Type == adSmallInt))
            {
                printf("Field: %s\n" ,(LPCSTR)fldTemp->
                    GetItem(Index)->GetName());
                printf("Numeric scale: %d\n", fldTemp->
                    GetItem(Index)->GetNumericScale());
                printf("Precision: %d\n", fldTemp->
                    GetItem(Index)->GetPrecision());
            }
        }
    }
    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        // Pass a connection pointer accessed from the Recordset.
        _variant_t vtConnect = pRstDiscounts->GetActiveConnection();

        // GetActiveConnection returns connect string if connection
    }
}

```



```

//      PrintComError Function                                     //
//                                                                 //
//////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndNumericScaleCpp

```

See Also

[NumericScale Property](#) | [Precision Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Open and Close Methods Example (VC++)

This example uses the **Open** and [Close](#) methods on both [Recordset](#) and [Connection](#) objects that have been opened.

```
// BeginOpenCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <oledb.h>
#include <stdio.h>
#include <conio.h>
#include "OpenX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void OpenX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    OpenX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//      OpenX Function                                                //
//                                                                    //
////////////////////////////////////

void OpenX(void)
```

```

{
// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADO:: namespace
_RecordsetPtr pRstEmployee = NULL;
_ConnectionPtr pConnection = NULL;

// Define string variables.
_bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
    "Initial Catalog='pubs';Integrated Security='SSPI'");

// Define Other Variables.
HRESULT hr = S_OK;
IADORecordBinding *picRs = NULL; // Interface Pointer declar
CEmployeeRs emprs; // C++ Class object
DBDATE varDate;

try
{
// open connection and record set
TESTHR(pConnection.CreateInstance(__uuidof(Connection));
pConnection->Open(strCnn, "", "", adConnectUnspecified);

TESTHR(pRstEmployee.CreateInstance(__uuidof(Recordset));
pRstEmployee->Open("Employee",
    _variant_t((IDispatch *)pConnection,true), adOpenKeyset,
    adLockOptimistic, adCmdTable);

// Open an IADORecordBinding interface pointer which we'll
// use for Binding Recordset to a class.
TESTHR(pRstEmployee->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

//Bind the Recordset to a C++ Class here.
TESTHR(picRs->BindToRecordset(&emprs));

// Assign the first employee record's hire date
// to a variable, then change the hire date.
varDate = emprs.m_size_hiredate;
printf("\nOriginal data\n");
printf("\tName - Hire Date\n");
printf(" %s %s - %d/%d/%d\n\n",
    emprs.le_fnameStatus == adFldOK ?
    emprs.m_size_fname : "<NULL>",
    emprs.le_lnameStatus == adFldOK ?
    emprs.m_size_lname : "<NULL>",
    emprs.le_hiredateStatus == adFldOK ?
    emprs.m_size_hiredate.month : 0,
    emprs.le_hiredateStatus == adFldOK ?
    emprs.m_size_hiredate.day : 0,

```

```

        emprs.le_hiredateStatus == adFldOK ?
        emprs.m_size_hiredate.year : 0);

    emprs.m_size_hiredate.year=1900;
    emprs.m_size_hiredate.month=1;
    emprs.m_size_hiredate.day=1;
    picRs->Update(&empRs);

    printf("\nChanged data\n");
    printf("\tName - Hire Date\n");
    printf(" %s %s - %d/%d/%d\n\n",
        emprs.le_fnameStatus == adFldOK ?
        emprs.m_size_fname : "<NULL>",
        emprs.le_lnameStatus == adFldOK ?
        emprs.m_size_lname : "<NULL>",
        emprs.le_hiredateStatus == adFldOK ?
        emprs.m_size_hiredate.month : 0,
        emprs.le_hiredateStatus == adFldOK ?
        emprs.m_size_hiredate.day : 0,
        emprs.le_hiredateStatus == adFldOK ?
        emprs.m_size_hiredate.year : 0);

    // Requery Recordset and reset the hire date.
    pRstEmployee->Requery(adOptionUnspecified);
    // Open an IADORecordBinding interface pointer which we'll
    // use for Binding Recordset to a class.
    TESTHR(pRstEmployee->QueryInterface(
        __uuidof(IADORecordBinding), (LPVOID*)&picRs));

    // Rebind the Recordset to a C++ Class here.
    TESTHR(picRs->BindToRecordset(&empRs));
    emprs.m_size_hiredate = varDate;
    picRs->Update(&empRs);
    printf("\nData after reset\n");
    printf("\tName - Hire Date\n");
    printf(" %s %s - %d/%d/%d", emprs.le_fnameStatus == adFldOK
        emprs.m_size_fname : "<NULL>",
        emprs.le_lnameStatus == adFldOK ?
        emprs.m_size_lname : "<NULL>",
        emprs.le_hiredateStatus == adFldOK ?
        emprs.m_size_hiredate.month : 0,
        emprs.le_hiredateStatus == adFldOK ?
        emprs.m_size_hiredate.day : 0,
        emprs.le_hiredateStatus == adFldOK ?
        emprs.m_size_hiredate.year : 0);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.

```

```

        // Pass a connection pointer accessed from the Connection.
        PrintProviderError(pConnection);
        PrintComError(e);
    }

    // Clean up objects before exit.
    if (pRstEmployee)
        if (pRstEmployee->State == adStateOpen)
            pRstEmployee->Close();
    if (pConnection)
        if (pConnection->State == adStateOpen)
            pConnection->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                     //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());
}

```

```

    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndOpenCpp

```

OpenX.h

```

// BeginOpenH
#include "icrsint.h"

// This Class extracts only fname,lastname and
// hire_date from employee table
class CEmployeeRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

    // Column fname is the 2nd field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_size_fname,
        sizeof(m_size_fname), le_fnameStatus, FALSE)

    // Column lname is the 4th field in the table.
    ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_size_lname,
        sizeof(m_size_lname), le_lnameStatus, FALSE)

    // Column hiredate is the 8th field in the table.
    ADO_VARIABLE_LENGTH_ENTRY2(8, adDBDate, m_size_hiredate,
        sizeof(m_size_hiredate), le_hiredateStatus, TRUE)

END_ADO_BINDING()

public:
    CHAR        m_size_fname[21];
    ULONG       le_fnameStatus;
    CHAR        m_size_lname[31];
    ULONG       le_lnameStatus;
    DBDATE      m_size_hiredate;
    ULONG       le_hiredateStatus;
};
// EndOpenH

```

See Also

[Close Method](#) | [Connection Object](#) | [Open Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

OpenSchema Method Example (VC++)

This example uses the [OpenSchema](#) method to display the name and type of each table in the *Pubs* database.

```
// BeginOpenSchemaCpp
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <oleauto.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void OpenSchemaX(void);
void OpenSchemaX2(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

//////////////////////////////////////
//                                     //
//      Main Function                   //
//                                     //
//////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    OpenSchemaX();

    printf("Press any key to see the results of 2nd "
        "function...\n\n");
    getch();

    OpenSchemaX2();

    ::CoUninitialize();
}
```

```

////////////////////////////////////
//                                                                    //
//      OpenSchemaX Function                                          //
//                                                                    //
////////////////////////////////////

void OpenSchemaX()
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _ConnectionPtr pConnection    = NULL;
    _RecordsetPtr  pRstSchema    = NULL;

    //Other Variables
    HRESULT hr = S_OK;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        pConnection->Open (strCnn, "", "", adConnectUnspecified);

        pRstSchema = pConnection->OpenSchema(adSchemaTables);

        while(!(pRstSchema->EndOfFile))
        {
            _bstr_t table_name = pRstSchema->Fields->
                GetItem("TABLE_NAME")->Value;

            printf("Table Name: %s\n", (LPCSTR) table_name);

            _bstr_t table_type = pRstSchema->Fields->
                GetItem("TABLE_TYPE")->Value;

            printf("Table type: %s\n\n", (LPCSTR) table_type);

            pRstSchema->MoveNext();

            int intLine = intLine + 1;
            if (intLine % 5 == 0)
            {
                printf("\nPress any key to continue...");
                getch();
                //Clear the screen for the next display
                system("cls");
            }
        }
    }
}

```

```

    }
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Connection.
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
if (pRstSchema)
    if (pRstSchema->State == adStateOpen)
        pRstSchema->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

////////////////////////////////////
//                                                                    //
//      OpenSchemaX2 Function                                          //
//                                                                    //
////////////////////////////////////
void OpenSchemaX2()
{
    HRESULT hr = S_OK;
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _ConnectionPtr pConnection2 = NULL;
    _RecordsetPtr pRstSchema = NULL;

    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        // Open connection.
        TESTHR(pConnection2.CreateInstance(__uuidof(Connection));
        pConnection2->Open (strCnn, "", "", adConnectUnspecified);

        // Create a safearray which takes four elements, and pass it
        // 2nd parameter in OpenSchema method.
        SAFEARRAY FAR* psa = NULL;
        SAFEARRAYBOUND rgsabound;
        _variant_t var;
        _variant_t Array;
        rgsabound.lLbound = 0;
    }
}

```

```

rgsabound.cElements = 4;
psa = SafeArrayCreate(VT_VARIANT, 1, &rgsabound);
var.vt = VT_EMPTY;

long ix;
ix = 0;
SafeArrayPutElement(psa, &ix, &var);

ix= 1;
SafeArrayPutElement(psa, &ix, &var);

ix = 2;
SafeArrayPutElement(psa, &ix, &var);

var.vt = VT_BSTR;
char * s1 = "VIEW";
_bstr_t str = s1;
var.bstrVal = str;

ix = 3;
SafeArrayPutElement(psa, &ix, &var);

Array.vt = VT_ARRAY|VT_VARIANT;
Array.parray = psa;

pRstSchema = pConnection2->OpenSchema(adSchemaTables,&Array)

while(!(pRstSchema->EndOfFile))
{
    printf("Table Name: %s\n", (LPCSTR) (_bstr_t) pRstSchema
        Fields->GetItem("TABLE_NAME")->Value);

    printf("Table type: %s\n\n", (LPCSTR) (_bstr_t) pRstSchem
        Fields->GetItem("TABLE_TYPE")->Value);

    pRstSchema->MoveNext();

    int intLine = intLine + 1;
    if (intLine % 5 == 0)
    {
        printf("\nPress any key to continue...");
        getch();
        //Clear the screen for the next display
        system("cls");
    }
}
} // End Try statement.
catch (_com_error &e)
{
    // Notify the user of errors if any.

```

```

        // Pass a connection pointer accessed from the Connection.
        PrintProviderError(pConnection2);
        PrintComError(e);
    }

    // Clean up objects before exit.
    if (pRstSchema)
        if (pRstSchema->State == adStateOpen)
            pRstSchema->Close();
    if (pConnection2)
        if (pConnection2->State == adStateOpen)
            pConnection2->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());
}

```

```
    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndOpenSchemaCpp
```

See Also

[OpenSchema Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Optimize Property Example (VC++)

This example demonstrates the [Field](#) object [dynamic Optimize](#) property. The *zip* field of the *Authors* table in the *Pubs* database is not indexed. Setting the [Optimize](#) property to **True** on the *zip* field authorizes ADO to build an index that improves the performance of the [Find](#) method.

```
// BeginOptimizeCpp
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void OptimizeX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

/////////////////////////////////////////////////////////////////
//                                                                    //
//      Main Function                                                    //
//                                                                    //
/////////////////////////////////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    OptimizeX();

    ::CoUninitialize();
}

/////////////////////////////////////////////////////////////////
//                                                                    //
//      OptimizeX Function                                              //
//                                                                    //
/////////////////////////////////////////////////////////////////
void OptimizeX(void)
{
    HRESULT    hr = S_OK;
```

```

// Define string variables.
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");
// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
_RecordsetPtr pRst = NULL;

try
{
    TESTHR(pRst.CreateInstance(__uuidof(Recordset)));

    // Enable Index creation.
    pRst->CursorLocation = adUseClient;
    pRst->Open ("SELECT * FROM authors", strCnn,
              adOpenStatic, adLockReadOnly, adCmdText);

    // Create the index
    pRst->Fields->GetItem("zip")->Properties->
        GetItem("Optimize")->PutValue("True");

    // Find Akiko Yokomoto
    pRst->Find("zip = '94595'",1,adSearchForward);
    printf("\n%s %s %s %s %s\n",
          (LPSTR) (_bstr_t) pRst->Fields->GetItem("au_fname")->Val
          (LPSTR) (_bstr_t) pRst->Fields->GetItem("au_lname")->Val
          (LPSTR) (_bstr_t) pRst->Fields->GetItem("address")->Valu
          (LPSTR) (_bstr_t) pRst->Fields->GetItem("city")->Value,
          (LPSTR) (_bstr_t) pRst->Fields->GetItem("state")->Value)

    // Delete the index
    pRst->Fields->GetItem("zip")->Properties->
        GetItem("Optimize")->PutValue("False");
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRst->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
    }
}

```

```

        default:
            printf("Errors occurred.");
            break;
    }
}

// Clean up objects before exit.
if (pRst)
    if (pRst->State == adStateOpen)
        pRst->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                     //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print COM errors.
    printf("Error\n");
}

```

```
printf("\tCode = %08lx\n", e.Error());  
printf("\tCode meaning = %s\n", e.ErrorMessage());  
printf("\tSource = %s\n", (LPCSTR) bstrSource);  
printf("\tDescription = %s\n", (LPCSTR) bstrDescription);  
}  
// EndOptimizeCpp
```

See Also

[Field Object | Optimize Property—Dynamic \(ADO\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

OriginalValue and UnderlyingValue Properties Example (VC++)

This example demonstrates the [OriginalValue](#) and [UnderlyingValue](#) properties by displaying a message if a record's underlying data has changed during a [Recordset](#) batch update.

```
// BeginOriginalValueCpp
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include "OriginalValueX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void OriginalValueX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                               //
//      Main Function             //
//                               //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    OriginalValueX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//      OriginalValueX Function  //
//                               //
////////////////////////////////////
```

```

void OriginalValueX(void)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _ConnectionPtr    pConnection    = NULL;
    _FieldPtr         pFldType       = NULL;
    _RecordsetPtr     pRstTitles     = NULL;

    // Define string variables.
    _bstr_t strSQLChange("UPDATE Titles SET Type = "
        "'sociology' WHERE Type = 'psychology'");
    _bstr_t strSQLRestore("UPDATE Titles SET Type = "
        "'psychology' WHERE Type = 'sociology'");

    // Define Other Variables
    HRESULT    hr = S_OK;
    IADORecordBinding    *picRs = NULL; //Interface Pointer declare
    CTitleRs titlers;    // C++ Class object

    try
    {
        _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer'
            'Initial Catalog='pubs';Integrated Security='SSPI'");

        // Open connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
        pConnection->Open (strCnn, "", "", adConnectUnspecified);

        // Open Recordset for batch update.
        TESTHR(pRstTitles.CreateInstance(__uuidof(Recordset)));
        pRstTitles->PutActiveConnection(
            _variant_t((IDispatch *)pConnection,true));
        pRstTitles->CursorType = adOpenKeyset;
        pRstTitles->LockType = adLockBatchOptimistic;

        // Cast Connection pointer to an IDispatch type so converted
        // to correct type of variant.
        pRstTitles->Open("Titles",
            _variant_t((IDispatch *)pConnection,true),
            adOpenKeyset, adLockBatchOptimistic, adCmdTable);

        //Open an IADORecordBinding interface pointer which
        //we'll use for Binding Recordset to a class.
        TESTHR(pRstTitles->QueryInterface(
            __uuidof(IADORecordBinding), (LPVOID*)&picRs));

        //Bind the Recordset to a C++ Class here
        TESTHR(picRs->BindToRecordset(&titlers));
    }
}

```

```

// Set field object variable for Type field.
pFldType = pRstTitles->Fields->GetItem("type");

// Change the type of psychology titles.
while(!(pRstTitles->EndOfFile))
{
    if (!strcmp(strtok((char *)titlers.m_szau_Type, " "),
                "psychology"))
    {
        pFldType->Value = "self_help";
    }
    pRstTitles->MoveNext();
}

// Simulate a change by another user by updating data
// using a command string.
pConnection->Execute(strSQLChange, NULL, 0);

// Check for changes.
pRstTitles->MoveFirst();
while(!(pRstTitles->EndOfFile))
{
    if (strcmp(pFldType->OriginalValue.pcVal,
                pFldType->UnderlyingValue.pcVal))
    {
        printf("\n\nData has changed!");

        printf("\n\nTitle ID: %s", titlers.lau_Title_idStatus
                adFldOK ? titlers.m_szau_Title_id : "<NULL>");

        printf("\n\nCurrent Value: %s",
                (LPCSTR) (_bstr_t) pFldType->Value);

        printf("\n\nOriginal Value: %s",
                (LPCSTR) (_bstr_t) pFldType->OriginalValue);

        printf("\n\nUnderlying Value: %s\n\n",
                (LPCSTR) (_bstr_t) pFldType->UnderlyingValue);

        printf("Press any key to continue...");
        getch();

        system("cls");
    }
    pRstTitles->MoveNext();
}
}
catch (_com_error &e)

```

```

    {
        // Notify the user of errors if any.
        // Pass a connection pointer accessed from the Connection.
        PrintProviderError(pConnection);
        PrintComError(e);
    }

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstTitles)
    if (pRstTitles->State == adStateOpen)
    {
        // Cancel the update because this is a demonstration.
        pRstTitles->CancelBatch(adAffectAll);
        pRstTitles->Close();
    }
if (pConnection)
    if (pConnection->State == adStateOpen)
    {
        // Restore Original Values.
        pConnection->Execute(strSQLRestore, NULL, 0);
        pConnection->Close();
    }
};

//////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
//////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

```

```

    }
}

////////////////////////////////////
//
//      PrintComError Function
//
////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndOriginalValueCpp

```

OriginalValueX.h

```

// BeginOriginalValueH
#include "icrsint.h"

//This class extracts title_id and type from titles table
class CTitlerS : public CAD0RecordBinding
{
BEGIN_ADO_BINDING(CTitlerS)
    // Column title_id is the 1st field in the Recordset
    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_szau_Title_id,
        sizeof(m_szau_Title_id), lau_Title_idStatus, FALSE)

    // Column type is the 3rd field in the Recordset
    ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_szau_Type,
        sizeof(m_szau_Type), lau_TypeStatus, TRUE)
END_ADO_BINDING()

public:
    CHAR m_szau_Title_id[7];
    ULONG lau_Title_idStatus;
    CHAR m_szau_Type[13];
    ULONG lau_TypeStatus;
};
// EndOriginalValueH

```

See Also

[OriginalValue Property](#) | [Recordset Object](#) | [UnderlyingValue Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Prepared Property Example (VC++)

This example demonstrates the [Prepared](#) property by opening two [Command](#) objects—one prepared and one not prepared.

```
// BeginPreparedCpp
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include <winbase.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void PreparedX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

/////////////////////////////////////////////////////////////////
//
//      Main Function
//
/////////////////////////////////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    PreparedX();

    ::CoUninitialize();
}

/////////////////////////////////////////////////////////////////
//
//      PreparedX Function
//
/////////////////////////////////////////////////////////////////

void PreparedX(void)
{
    // Define ADO object pointers.
```

```

// Initialize pointers on define.
// These are in the ADODB:: namespace.
_ConnectionPtr  pConnection  =NULL;
_CommandPtr     pCmd1       =NULL;
_CommandPtr     pCmd2       =NULL;

// Define string variables.
_bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

//Define Other Variables
HRESULT  hr = S_OK;

try
{
    // Open a connection.
    TESTHR(pConnection.CreateInstance(__uuidof(Connection));
    pConnection->Open (strCnn, "", "", adConnectUnspecified);

    _bstr_t strCmd ("SELECT title,type FROM titles ORDER BY type

// Create two command objects for the same
// command - one prepared and one not prepared.
    TESTHR(pCmd1.CreateInstance(__uuidof(Command)));
    pCmd1->ActiveConnection = pConnection;
    pCmd1->CommandText = strCmd;

    TESTHR(pCmd2.CreateInstance(__uuidof(Command)));
    pCmd2->ActiveConnection = pConnection;
    pCmd2->CommandText = strCmd;
    pCmd2->PutPrepared(true);

// Set a timer,then execute the unprepared command 20 times.
    DWORD sngStart=GetTickCount();

    for(int intLoop=1;intLoop<=20;intLoop++)
    {
        pCmd1->Execute(NULL, NULL, adCmdText);
    }
    DWORD sngEnd=GetTickCount();

    float sngNotPrepared = (float)(sngEnd - sngStart)/(float)100

// Reset the timer,then execute the prepared command 20 time
    sngStart=GetTickCount();
    for(intLoop=1;intLoop<=20;intLoop++)
    {
        pCmd2->Execute(NULL, NULL, adCmdText);
    }
    sngEnd=GetTickCount();
}

```



```
//
////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndPreparedCpp
```

See Also

[Command Object](#) | [Prepared Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

{
    HRESULT    hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _ConnectionPtr    pConnection1    = NULL;
    _ConnectionPtr    pConnection2    = NULL;
    _ConnectionPtr    pConnection3    = NULL;

    try
    {
        // Open a Connection using the Microsoft ODBC provider.
        TESTHR(pConnection1.CreateInstance(__uuidof(Connection)));
        pConnection1->ConnectionString = "driver={SQL Server};"
            "server='MySqlServer';user id='MyUserId';password='MyPas
pConnection1->Open("", "", "", adConnectUnspecified);
pConnection1->DefaultDatabase = "pubs";

        // Display the provider
        printf("\n\nConnection1 provider: %s \n\n",
            (LPCSTR)pConnection1->Provider);

        // Open a connection using the OLE DB Provider for Microsoft
        TESTHR(pConnection2.CreateInstance(__uuidof(Connection)));
        pConnection2->Provider = "Microsoft.Jet.OLEDB.4.0";

        char *sConn = "c:\\Program Files\\Microsoft Office\\Office\\
            "Samples\\Northwind.mdb";

        pConnection2->Open(sConn, "admin", "", NULL);

        // Display the provider
        printf("Connection2 provider: %s \n\n", (LPCSTR)pConnection2-
            Provider);

        // Open a Connection using the Microsoft SQL Server provider
        TESTHR(pConnection3.CreateInstance(__uuidof(Connection)));
        pConnection3->Provider = "sqloledb";
        pConnection3->Open("Data Source='MySqlServer';Initial Catalo
            "MyUserId", "MyPassword", NULL);

        // Display the provider.
        printf("Connection3 provider: %s\n\n", (LPCSTR)pConnection3->
            Provider);
    }

    catch (_com_error &e)
    {
        // Notify the user of errors if any.
    }
}

```

```

        PrintProviderError(pConnection1);
        if(pConnection2) PrintProviderError(pConnection2);
        if(pConnection3) PrintProviderError(pConnection3);
        PrintComError(e);
    }

    if (pConnection1)
        if (pConnection1->State == adStateOpen)
            pConnection1->Close();
    if (pConnection2)
        if (pConnection2->State == adStateOpen)
            pConnection2->Close();
    if (pConnection3)
        if (pConnection3->State == adStateOpen)
            pConnection3->Close();
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                             //
//                                                                                               //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                 //
//                                                                                               //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{

```

```
_bstr_t bstrSource(e.Source());
_bstr_t bstrDescription(e.Description());

// Print COM errors.
printf("Error\n");
printf("\tCode = %08lx\n", e.Error());
printf("\tCode meaning = %s\n", e.ErrorMessage());
printf("\tSource = %s\n", (LPCSTR) bstrSource);
printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndProviderCpp
```

See Also

[Connection Object](#) | [DefaultDatabase Property](#) | [Provider Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Refresh Method Example (VC++)

This example demonstrates using the [Refresh](#) method to refresh the [Parameters](#) collection for a stored procedure [Command](#) object.

```
// BeginRefreshCpp
#import "c:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include "RefreshX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void RefreshX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);
inline char* mygets(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return NULL;

    if (!strrchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strrchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strrchr(strDest, '\n') - strDest] = '\0';

    return pstrRet;
}

////////////////////////////////////
//                               //
//   Main Function                //
//                               //
////////////////////////////////////
```

```

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    RefreshX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                     //
//          RefreshX Function          //
//                                     //
////////////////////////////////////

void RefreshX(void)
{
    HRESULT hr = S_OK;

    // Define string variables.
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _ConnectionPtr pConnection = NULL;
    _CommandPtr pCmdByRoyalty = NULL;
    _RecordsetPtr pRstByRoyalty = NULL;
    _RecordsetPtr pRstAuthors = NULL;
    IADORecordBinding *picRS = NULL; //Interface Pointer declara
    CAutorsRs authorsrs; //C++ class object

    try
    {
        // Open connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        pConnection->Open (strCnn, "", "", adConnectUnspecified);

        // Open a command object for a stored procedure,
        // with one parameter.
        TESTHR(pCmdByRoyalty.CreateInstance(__uuidof(Command));
        pCmdByRoyalty->ActiveConnection = pConnection;
        pCmdByRoyalty->CommandText = "byroyalty";
        pCmdByRoyalty->CommandType = adCmdStoredProc;
        pCmdByRoyalty->Parameters->Refresh();

        // Get parameter value and execute the command,
        // storing the results in a recordset.
    }
}

```

```

char *strRoyalty;
char strTemp[5];
do
{
    printf("\n\nEnter royalty : ");
    mygets(strTemp, 5);

    strRoyalty = strtok(strTemp, " \t");
    if(strRoyalty == NULL)
    {
        exit(1);
    }

    // if the input is not numeric then notify the user.
    if(!atoi(strRoyalty))
    {
        printf("\nExpecting numeric value...");
        continue;
    }
}while(!atoi(strRoyalty));

_variant_t vtroyal;
vtroyal.vt = VT_I2;
vtroyal.iVal = atoi(strRoyalty);
_variant_t Index;
Index.vt = VT_I2;
Index.iVal = 1;
pCmdByRoyalty->GetParameters()->GetItem(Index)->
    PutValue(vtroyal);
pRstByRoyalty = pCmdByRoyalty->
    Execute(NULL, NULL, adCmdStoredProc);

// Open the Authors table to get author names for display.
TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));
pRstAuthors->Open ("authors",
    _variant_t((IDispatch *) pConnection, true),
    adOpenForwardOnly, adLockReadOnly, adCmdTable);

//Open an IADORecordBinding interface pointer which we'll us
//Binding Recordset to a class.
TESTHR(pRstAuthors->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

//Bind the Recordset to a C++ Class here.
TESTHR(picRs->BindToRecordset(&authorsrs));

// Print current data in the recordset, adding
// author names from Authors table.
printf("\n\nAuthors with %s percent royalty\n\n", strRoyalty)

```

```

while(!(pRstByRoyalty->EndOfFile))
{
    _bstr_t strAuthorID = pRstByRoyalty->Fields->GetItem(
        "au_id")->Value;

    printf(" %s", (LPCSTR) (_bstr_t) pRstByRoyalty->Fields->
        GetItem("au_id")->Value);

    pRstAuthors->Filter = "au_id='"+strAuthorID+"'";
    printf(", %s %s\n\n", authorsrs.lau_fnameStatus == adFld0
        authorsrs.m_szau_fname : "<NULL>",
        authorsrs.lau_lnameStatus == adFldOK ?
        authorsrs.m_szau_lname : "<NULL>");
    pRstByRoyalty->MoveNext();
}
}
catch (_com_error &e)
{
    PrintProviderError(pConnection);
    PrintComError(e);
}

if (pRstByRoyalty)
    if (pRstByRoyalty->State == adStateOpen)
        pRstByRoyalty->Close();
if (pRstAuthors)
    if (pRstAuthors->State == adStateOpen)
        pRstAuthors->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                     //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(long i = 0;i < nCount;i++)

```

```

        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s", pErr->Number,
                pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndRefreshCpp

```

RefreshX.h

```

// BeginRefreshH
#include "icrsint.h"

//This Class extracts lname, fname from authors table.
class CAutorsRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CAutorsRs)
    // Column lname is the 2nd field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szau_lname,
        sizeof(m_szau_lname), lau_lnameStatus, TRUE)
    // Column fname is the 3rd field in the recordset.
    ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_szau_fname,
        sizeof(m_szau_fname), lau_fnameStatus, TRUE)

END_ADO_BINDING()

public:
    CHAR m_szau_fname[21];
    ULONG lau_fnameStatus;
    CHAR m_szau_lname[41];

```

```
        ULONG   lau_lnameStatus;  
};  
// EndRefreshH
```

See Also

[Command Object](#) | [Parameters Collection](#) | [Refresh Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Resync Method Example (VC++)

This example demonstrates using the [Resync](#) method to refresh data in a static recordset.

```
// BeginResyncCpp
#import "c:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void ResyncX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                               //
//   Main Function               //
//                               //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    ResyncX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//   ResyncX Function           //
//                               //
////////////////////////////////////

void ResyncX(void)
{
    HRESULT hr = S_OK;
```

```

// Define string variables.
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
_RecordsetPtr pRstTitles = NULL;

try
{
    // Open recordset for titles table.
    TESTHR(pRstTitles.CreateInstance(__uuidof(Recordset)));
    pRstTitles->CursorLocation = adUseClient;
    pRstTitles->CursorType = adOpenStatic;
    pRstTitles->LockType = adLockBatchOptimistic;
    pRstTitles->Open ("titles",strCnn,
                    adOpenStatic, adLockBatchOptimistic, adCmdTable);

    // Change the type of the first title in the recordset.
    pRstTitles->Fields->GetItem("type")->Value =
        (_bstr_t) ("database");

    // Display the results of the change.
    printf("\nBefore resync: \n\n");

    printf("Title - %s\n\n", (LPSTR) (_bstr_t) pRstTitles->
           Fields->GetItem("title")->Value);

    printf("Type - %s\n\n", (LPSTR) (_bstr_t) pRstTitles->
           Fields->GetItem("type")->Value);

    // Resync with database.
    pRstTitles->Resync(adAffectAll, adResyncAllValues);

    // Display the results of the resynch.
    printf("\n\nAfter resync: \n\n");

    printf("Title - %s\n\n", (LPSTR) (_bstr_t) pRstTitles->
           Fields->GetItem("title")->Value);

    printf("Type - %s\n\n", (LPSTR) (_bstr_t) pRstTitles->
           Fields->GetItem("type")->Value);
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstTitles->GetActiveConnection();

```

```

// GetActiveConnection returns connect string if connection
// is not open, else returns Connection object.
switch(vtConnect.vt)
{
    case VT_BSTR:
        PrintComError(e);
        break;
    case VT_DISPATCH:
        PrintProviderError(vtConnect);
        break;
    default:
        printf("Errors occured.");
        break;
}
}

if (pRstTitles)
    if (pRstTitles->State == adStateOpen)
    {
        pRstTitles->CancelBatch(adAffectAll);
        pRstTitles->Close();
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                             //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("\t Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}
}

```

```

////////////////////////////////////
//                                                                    //
//      PrintComError Function                                        //
//                                                                    //
////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndResyncCpp

```

See Also

[Resync Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Save and Open Methods Example (VC++)

These three examples demonstrate how the [Save](#) and **Open** methods can be used together.

Assume you are going on a business trip and want to take along a table from a database. Before you go, you access the data as a [Recordset](#) and save it in a transportable form. When you arrive at your destination, you access the **Recordset** as a local, disconnected **Recordset**. You make changes to the **Recordset**, then save it again. Finally, when you return home, you connect to the database again and update it with the changes you made on the road.

```
// BeginSaveCpp
#import "c:\Program Files\Common Files\system\ado\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
bool FileExists(void);
void SaveX1(void);
void SaveX2(void);
void SaveX3(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//          Main Function                                           //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    //If File exists in the specified directory, then display error
```

```

    if (!FileExists())
    {
        SaveX1();
        SaveX2();
        SaveX3();
    }

    ::CoUninitialize();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      SaveX1 Function                                                                           //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//First, access and save the authors table.
void SaveX1()
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr pRstAuthors = NULL;

    //Definitions of other variables
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    try
    {
        TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));

        pRstAuthors->Open("SELECT * FROM authors",strCnn,
            adOpenDynamic,adLockBatchOptimistic,adCmdText);

        // For sake of illustration, save the Recordset to a diskett
        // in XML format.
        pRstAuthors->Save("c:\\pubs.xml",adPersistXML);
    }
    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        // Pass a connection pointer accessed from the Recordset.
        _variant_t vtConnect = pRstAuthors->GetActiveConnection();

        // GetActiveConnection returns connect string if connection
        // is not open, else returns Connection object.
        switch(vtConnect.vt)

```

```

        {
            case VT_BSTR:
                PrintComError(e);
                break;
            case VT_DISPATCH:
                PrintProviderError(vtConnect);
                break;
            default:
                printf("Errors occurred.");
                break;
        }
    }

    if (pRstAuthors)
        if (pRstAuthors->State == adStateOpen)
            pRstAuthors->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//          SaveX2 Function                                                                                   //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//At this point, you have arrived at your destination.
//You will access the authors table as a local, disconnected Records
//Don't forget you must have the MSPersist provider on the machine y
//are using in order to access the saved file, c:\pubs.xml.
void SaveX2()
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr pRstAuthors = NULL;

    try
    {
        TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));

        //For sake of illustration, we specify all parameters.
        pRstAuthors->Open("c:\\pubs.xml", "Provider='MSPersist';",
            adOpenForwardOnly, adLockBatchOptimistic, adCmdFile);

        //Now you have a local, disconnected Recordset.
        //Edit it as you desire.
        //(In this example, the change makes no difference).
        pRstAuthors->Find("au_lname = 'Carson'", NULL, adSearchForward
            if (pRstAuthors->EndOfFile)

```

```

    {
        printf("Name not found ...\n");
        pRstAuthors->Close();
        return;
    }
    pRstAuthors->GetFields()->GetItem("City")->PutValue("Chicago");
    pRstAuthors->Update();

    // Save changes in ADTG format this time, purely for sake of
    // illustration. Note that the previous version is still on
    // diskette as c:\pubs.xml.
    pRstAuthors->Save("c:\\pubs.adtg",adPersistADTG);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstAuthors->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

if (pRstAuthors)
    if (pRstAuthors->State == adStateOpen)
        pRstAuthors->Close();
}

////////////////////////////////////
//                                                                    //
//   SaveX3 Function                                                    //
//                                                                    //
////////////////////////////////////
//Finally, you return home. Now update the database with
//your changes.
void SaveX3()
{
    HRESULT hr = S_OK;

```

```

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADO:: namespace.
_RecordsetPtr pRstAuthors = NULL;
_ConnectionPtr pCnn = NULL;

//Definitions of other variables
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

try
{
    TESTHR(pCnn.CreateInstance(__uuidof(Connection)));

    TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));

    //If there is no ActiveConnection, you can open with default
    pRstAuthors->Open("c:\\pubs.adtg", "Provider=MSPersist;",
                    adOpenForwardOnly, adLockBatchOptimistic, adCmdFile);

    //Connect to the database, associate the Recordset with
    //the connection, then update the database table with the
    //changed Recordset.
    pCnn->Open(strCnn, "", "", NULL);

    pRstAuthors->PutActiveConnection(_variant_t((IDispatch *) pC));
    pRstAuthors->UpdateBatch(adAffectAll);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstAuthors->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occurred.");
            break;
    }
}

```

```

}

if (pRstAuthors)
    if (pRstAuthors->State == adStateOpen)
        pRstAuthors->Close();
if (pCnn)
    if (pCnn->State == adStateOpen)
        pCnn->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                 //
//      PrintProviderError Function                                //
//                                                                 //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

        // Collection ranges from 0 to nCount -1.
        for(long i = 0;i < nCount;i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                 //
//      PrintComError Function                                    //
//                                                                 //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
}

```

```
printf("\tCode meaning = %s\n", e.ErrorMessage());
printf("\tSource = %s\n", (LPCSTR) bstrSource);
printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}

bool FileExists()
{
    struct _finddata_t xml_file;
    long hFile;

    if( (hFile = _findfirst("c:\\pubs.xml", &xml_file )) != -1L)
    {
        printf( "File already exists!\n" );
        return(true);
    }
    else
        return (false);
}
// EndSaveCpp
```

See Also

[Open Method \(ADO Recordset\)](#) | [Recordset Object](#) | [Save Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Seek Method and Index Property Example (VC++)

This example uses the [Recordset](#) object's [Seek](#) method and [Index](#) property in conjunction with a given *Employee ID*, to locate the employee's name in the *Employees* table of the Nwind.mdb database.

```
// BeginSeekCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <conio.h>
#include <string.h>
#include "SeekX.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void SeekX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);
inline char* mygets(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return NULL;

    if (!strrchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strrchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strrchr(strDest, '\n') - strDest] = '\0';

    return pstrRet;
}
```

```

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    SeekX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//      SeekX Function                                               //
//                                                                    //
////////////////////////////////////
void SeekX()
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.

    _RecordsetPtr pRstEmp = NULL;

    IADORecordBinding *picRs = NULL; // Interface Pointer declar
    CEmployeeRs EmpRs; //C++ class object

    //Definitions of other variables
    _bstr_t strPrompt("Enter an EmployeeID (e.g., 1 to 9)");
    char strEmpId[2];

    try
    {
        TESTHR(pRstEmp.CreateInstance(__uuidof(Recordset)));
        pRstEmp->CursorLocation = adUseServer;
        pRstEmp->Open("employees", "Provider='Microsoft.Jet.OLEDB.4.
            "Data Source='C:\\Program Files\\Microsoft Office\\Offic
            "Samples\\Northwind.mdb'";",
            adOpenKeyset,adLockReadOnly,adCmdTableDirect);

        //Open an IADORecordBinding interface pointer which
        //we'll use for binding Recordset to a Class
        TESTHR(pRstEmp->QueryInterface(
            __uuidof(IADORecordBinding), (LPVOID*)&picRs));
    }
}

```



```

        EmpRs.m_size_fname : "<NULL>",
        EmpRs.le_lnameStatus == adFldOK ?
        EmpRs.m_size_lname : "<NULL>");
    }
}
}
while(true);
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstEmp->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors ocured.");
            break;
    }
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstEmp)
    if (pRstEmp->State == adStateOpen)
        pRstEmp->Close();
}

////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                    //
//                                                                    //
////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.

```

```

ErrorPtr    pErr  = NULL;

if( (pConnection->Errors->Count) > 0)
{
    long nCount = pConnection->Errors->Count;

    // Collection ranges from 0 to nCount -1.
    for(long i = 0;i < nCount;i++)
    {
        pErr = pConnection->Errors->GetItem(i);
        printf("\t Error number: %x\t%s", pErr->Number,
            pErr->Description);
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print COM errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndSeekCpp

```

SeekX.h

```

// BeginSeekH
#include "icrsint.h"

// This Class extracts only EmployeeId,FirstName and LastName
// from employees table
class CEmployeeRs : public CADOResultBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

    // Column hiredate is the 1st field in the table.
    ADO_VARIABLE_LENGTH_ENTRY2(1, adInteger,m_ie_empid,
        sizeof(m_ie_empid), le_empidStatus, FALSE)

```

```
        // Column LastName is the 2nd field in the table.
        ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_size_lname,
            sizeof(m_size_lname), le_lnameStatus, FALSE)

        // Column FirstName is the 3rd field in the table.
        ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_size_fname,
            sizeof(m_size_fname), le_fnameStatus, FALSE)
    END_ADO_BINDING()

public:
    INT     m_ie_empid;
    ULONG   le_empidStatus;
    CHAR    m_size_fname[11];
    ULONG   le_fnameStatus;
    CHAR    m_size_lname[21];
    ULONG   le_lnameStatus;
};
// EndSeekH
```

See Also

[Index Property](#) | [Recordset Object](#) | [Seek Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Sort Property Example (VC++)

This example uses the [Recordset](#) object's [Sort](#) property to reorder the rows of a **Recordset** derived from the **Authors** table of the **Pubs** database. A secondary utility routine prints each row.

```
// BeginSortCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void SortX(void);
void SortXprint(_bstr_t title, _RecordsetPtr rstp);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//    Main Function                                                    //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    SortX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//    SortX Function                                                  //
//                                                                    //
////////////////////////////////////
void SortX(void)
{
    HRESULT hr = S_OK;
```

```

// Initialize pointers on define.
// These are in the ADODB:: namespace.
_ConnectionPtr pConnection      = NULL;
_RecordsetPtr  pRstAuthors     = NULL;

// Define string variables.
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

try
{
    TESTHR(pConnection.CreateInstance(__uuidof(Connection)));

    TESTHR(pRstAuthors.CreateInstance(__uuidof(Recordset)));

    pRstAuthors->CursorLocation = adUseClient;
    pConnection->Open (strCnn, "", "", adConnectUnspecified);
    pRstAuthors->Open("SELECT * FROM authors",
                    _variant_t((IDispatch *) pConnection),
                    adOpenStatic, adLockReadOnly, adCmdText);

    SortXprint("    Initial Order    ", pRstAuthors);

    //Clear the screen for the next display.
    printf("\nPress any key to continue...");
    getch();
    system("cls");

    pRstAuthors->Sort = "au_lname ASC, au_fname ASC";
    SortXprint("Last Name Ascending", pRstAuthors);

    //Clear the screen for the next display.
    printf("\nPress any key to continue...");
    getch();
    system("cls");

    pRstAuthors->Sort = "au_lname DESC, au_fname ASC";
    SortXprint("Last Name Descending", pRstAuthors);
}
catch(_com_error &e)
{
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
if (pRstAuthors)
    if (pRstAuthors->State == adStateOpen)
        pRstAuthors->Close();
if (pConnection)

```

```

        if (pConnection->State == adStateOpen)
            pConnection->Close();
    }

    ////////////////////////////////////////
    //                                     //
    //      SortXprint Function           //
    //                                     //
    ////////////////////////////////////////
    //This is the secondary utility routine that prints
    //the given title, and the contents of the specified Recordset.
    void SortXprint(_bstr_t title, _RecordsetPtr rstp)
    {
        printf("-----%s-----\n", (LPCSTR)title);
        printf("First Name  Last Name\n"
            "-----\n");
        rstp->MoveFirst();
        int intLineCnt = 4;
        while (!(rstp->EndOfFile))
        {
            _bstr_t aufname;
            _bstr_t aulname;
            aufname = rstp->GetFields()->GetItem("au_fname")->Value;
            aulname = rstp->GetFields()->GetItem("au_lname")->Value;
            printf("%s  %s\n", (LPCSTR) aufname, (LPCSTR) aulname);
            rstp->MoveNext();
            intLineCnt++;
            if (intLineCnt % 20 ==0)
                {
                    printf("\nPress any key to continue...\n");
                    getch();
                }
        }
    }

    ////////////////////////////////////////
    //                                     //
    //      PrintProviderError Function   //
    //                                     //
    ////////////////////////////////////////
    void PrintProviderError(_ConnectionPtr pConnection)
    {
        // Print Provider Errors from Connection object.
        // pErr is a record object in the Connection's Error collection.
        ErrorPtr    pErr = NULL;

        if( (pConnection->Errors->Count) > 0)
        {
            long nCount = pConnection->Errors->Count;

```

```

        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////
//                                                                    //
//  PrintComError Function                                           //
//                                                                    //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}

// EndSortCpp

```

See Also

[Recordset Object](#) | [Sort Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Source Property Example (VC++)

This example demonstrates the [Source](#) property by opening three [Recordset](#) objects based on different data sources.

```
// BeginSourceCpp
#import "c:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void SourceX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                               //
//   Main Function               //
//                               //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    SourceX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//   SourceX Function           //
//                               //
////////////////////////////////////
void SourceX(void)
{
    HRESULT hr = S_OK;

    // Define string variables.
    _bstr_t strCmdSQL("Select title,type,pubdate ")
```

```

    "FROM titles ORDER BY title");
_bstr_t strSQL("SELECT title_ID AS TitleID, title AS Title, "
    "publishers.pub_id AS PubID, pub_name AS PubName "
    "FROM publishers INNER JOIN titles "
    "ON publishers.pub_id = titles.pub_id "
    "ORDER BY Title");
_bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
    "Initial Catalog='pubs';Integrated Security='SSPI';");

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
_ConnectionPtr pConnection          = NULL;
_RecordsetPtr  pRstTitles           = NULL;
_RecordsetPtr  pRstPublishers       = NULL;
_RecordsetPtr  pRstPublishersDirect = NULL;
_RecordsetPtr  pRstTitlesPublishers = NULL;
_CommandPtr    pCmdSQL               = NULL;

try
{
    // Open a connection.
    TESTHR(pConnection.CreateInstance(__uuidof(Connection));
    pConnection->Open (strCnn, "", "", adConnectUnspecified);

    // Open a recordset based on a command object.
    TESTHR(pCmdSQL.CreateInstance(__uuidof(Command));
    pCmdSQL->ActiveConnection = pConnection;
    pCmdSQL->CommandText = strCmdSQL;
    pRstTitles = pCmdSQL->Execute(NULL, NULL, adCmdText);

    // Open a recordset based on a a table
    TESTHR(pRstPublishers.CreateInstance(__uuidof(Recordset));
    pRstPublishers->Open ("publishers",
        _variant_t((IDispatch *) pConnection, true),
        adOpenForwardOnly, adLockReadOnly, adCmdTable);

    // Open a recordset based on a table
    TESTHR(pRstPublishersDirect.CreateInstance(
        __uuidof(Recordset)));
    pRstPublishersDirect->Open ("publishers",
        _variant_t((IDispatch *) pConnection, true),
        adOpenForwardOnly, adLockReadOnly, adCmdTableDirect);

    // Open a recordset based on a SQL string.
    TESTHR(pRstTitlesPublishers.CreateInstance(
        __uuidof(Recordset)));
    pRstTitlesPublishers->Open(strSQL,
        _variant_t((IDispatch *) pConnection, true),
        adOpenForwardOnly, adLockReadOnly, adCmdText);
}

```

```

// Use the Source property to display the source of
// each recordset.
printf("rstTitles source: \n%s\n\n",
      (LPCSTR)(_bstr_t) pRstTitles->GetSource().bstrVal);
printf("rstPublishers source: \n%s\n\n",
      (LPCSTR)(_bstr_t) pRstPublishers->GetSource().bstrVal);
printf("rstPublishersDirect source: \n%s\n\n",
      (LPCSTR)(_bstr_t) pRstPublishersDirect->GetSource().bstrVal);
printf("rstTitlesPublishers source: \n%s\n\n",
      (LPCSTR)(_bstr_t) pRstTitlesPublishers->GetSource().bstrVal);
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    PrintProviderError(pConnection);
    PrintComError(e);
}

if (pRstTitles)
    if (pRstTitles->State == adStateOpen)
        pRstTitles->Close();
if (pRstPublishers)
    if (pRstPublishers->State == adStateOpen)
        pRstPublishers->Close();
if (pRstPublishersDirect)
    if (pRstPublishersDirect->State == adStateOpen)
        pRstPublishersDirect->Close();
if (pRstTitlesPublishers)
    if (pRstTitlesPublishers->State == adStateOpen)
        pRstTitlesPublishers->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//    PrintProviderError Function                                     //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {

```

```

    long nCount = pConnection->Errors->Count;
    // Collection ranges from 0 to nCount -1.
    for(long i = 0; i < nCount; i++)
    {
        pErr = pConnection->Errors->GetItem(i);
        printf("Error number: %x\t%s\n", pErr->Number,
            (LPCSTR) pErr->Description);
    }
}

////////////////////////////////////
//                                                                    //
//    PrintComError Function                                          //
//                                                                    //
////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndSourceCpp

```

See Also

[Recordset Object](#) | [Source Property \(ADO Recordset\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

State Property Example (VC++)

This example uses the [State](#) property to display a message while [asynchronous](#) connections are opening and asynchronous commands are executing.

```
// BeginStateCpp
#import "c:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void StateX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                     //
//   Main Function                     //
//                                     //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    StateX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                     //
//   StateX Function                   //
//                                     //
////////////////////////////////////
void StateX(void)
{
    HRESULT hr = S_OK;

    // Define string variables.
    _bstr_t strSQLChange("UPDATE Titles SET Type = "
```

```

        "'self_help' WHERE Type = 'psychology'");
_bstr_t strSQLRestore("UPDATE Titles SET Type = "
        "'psychology' WHERE Type = 'self_help'");
_bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
_ConnectionPtr    pConnection        = NULL;
_ConnectionPtr    pConnection2       = NULL;
_CommandPtr       pCmdChange         = NULL;
_CommandPtr       pCmdRestore        = NULL;

try
{
    // Open two asynchronous connections, displaying
    // a message while connecting.
    TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
    TESTHR(pConnection2.CreateInstance(__uuidof(Connection)));

    pConnection->Open (strCnn, "", "", adAsyncConnect);
    while(pConnection->State == adStateConnecting)
    {
        printf("Opening first connection....\n\n");
    }

    pConnection2->Open (strCnn, "", "", adAsyncConnect);
    while(pConnection2->State == adStateConnecting)
    {
        printf("Opening second connection....\n\n");
    }

    // Create two command objects.
    TESTHR(pCmdChange.CreateInstance(__uuidof(Command)));
    pCmdChange->ActiveConnection = pConnection;
    pCmdChange->CommandText = strSQLChange;

    TESTHR(pCmdRestore.CreateInstance(__uuidof(Command)));
    pCmdRestore->ActiveConnection = pConnection2;
    pCmdRestore->CommandText = strSQLRestore;

    // Executing the commands, displaying a message
    // while they are executing.
    pCmdChange->Execute(NULL, NULL, adAsyncExecute);
    while(pCmdChange->State == adStateExecuting)
    {
        printf("Change command executing...\n\n");
    }
}

```



```
////////////////////////////////////  
void PrintComError(_com_error &e)  
{  
    _bstr_t bstrSource(e.Source());  
    _bstr_t bstrDescription(e.Description());  
  
    // Print Com errors.  
    printf("Error\n");  
    printf("\tCode = %08lx\n", e.Error());  
    printf("\tCode meaning = %s\n", e.ErrorMessage());  
    printf("\tSource = %s\n", (LPCSTR) bstrSource);  
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);  
}  
// EndStateCpp
```

See Also

[State Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Status Property Example (VC++)

This example uses the [Status](#) property to display which records have been modified in a batch operation before a batch update has occurred.

```
// BeginStatusCpp
#import "c:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include "StatusX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void StatusX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//    Main Function                                                    //
//                                                                    //
////////////////////////////////////

void main()
{
    if(FAILED(CoInitialize(NULL)))
        return;

    StatusX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//    StatusX Function                                                //
//                                                                    //
////////////////////////////////////
void StatusX(void)
{
    HRESULT hr = S_OK;
```

```

// Define string variables.
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
IADORecordBinding *picRs    = NULL; // Interface Pointer Decla
CTitlers titlers;          // C++ Class Object
_RecordsetPtr pRstTitles    = NULL;
LPSTR p_TempStr             = NULL;

try
{
    // Open recordset for batch update
    TESTHR(hr = pRstTitles.CreateInstance(__uuidof(Recordset)));
    pRstTitles->CursorType = adOpenKeyset;
    pRstTitles->LockType = adLockBatchOptimistic;
    pRstTitles->Open ("titles", strCnn,
                    adOpenKeyset, adLockBatchOptimistic, adCmdTable);

    // Open an IADORecordBinding interface pointer which
    // we will use for binding Recordset to a class.
    TESTHR(hr = pRstTitles->QueryInterface(
        __uuidof(IADORecordBinding), (LPVOID*)&picRs));

    // Bind the Recordset to a C++ class here
    TESTHR(hr = picRs->BindToRecordset(&titlers));

    p_TempStr = (LPSTR) malloc(sizeof(titlers.m_szt_Type));

    // Change the type of psychology titles.
    while(!(pRstTitles->EndOfFile))
    {
        // Set the final character of the destination string to
        p_TempStr[sizeof(titlers.m_szt_Type)-1] = '\0';
        // The source string will get truncated if its length is
        // longer than the length of the destination string minu
        strncpy(p_TempStr, strtok(titlers.m_szt_Type, " "),
            sizeof(titlers.m_szt_Type)-1);

        // Compare the type of psychology titles
        if (!strcmp(p_TempStr, "psychology"))
        {
            // Copy "self_help" title field
            pRstTitles->Fields->GetItem("type")->Value =
                (_bstr_t) ("self_help");
        }
        pRstTitles->MoveNext();
    }
}

```

```

// Display Title ID and status
pRstTitles->MoveFirst();
while(!(pRstTitles->EndOfFile))
{
    if(pRstTitles->Status == adRecModified)
    {
        printf("%s - Modified\n",titlers.lt_Title_idStatus =
            adFldOK ? titlers.m_szt_Title_id : "<NULL>");
    }
    else
    {
        printf("%s\n",titlers.lt_Title_idStatus == adFldOK ?
            titlers.m_szt_Title_id : "<NULL>");
    }
    pRstTitles->MoveNext();
}
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstTitles->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occured.");
            break;
    }
}

// Deallocate the memory
if (p_TempStr)
    free(p_TempStr);

if (pRstTitles)
    if (pRstTitles->State == adStateOpen)
    {
        // Cancel the update because this is a demonstration.
        pRstTitles->CancelBatch(adAffectAll);
    }
}

```

```

        pRstTitles->Close();
    }
}

/////////////////////////////////////////////////////////////////
//                                                                    //
//    PrintProviderError Function                                     //
//                                                                    //
/////////////////////////////////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

/////////////////////////////////////////////////////////////////
//                                                                    //
//    PrintComError Function                                       //
//                                                                    //
/////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndStatusCpp

```

StatusX.h

```

// BeginStatusH
#include "icrsint.h"

//This class extracts title_id and type from titles table.
class CTitleRs : public CADORecordBinding
{
    BEGIN_ADO_BINDING(CTitleRs)
        // Column title_id is the 1st field in the table
        ADO_VARIABLE_LENGTH_ENTRY2(1,adVarChar,m_szt_Title_id,
            sizeof(m_szt_Title_id),lt_Title_idStatus,FALSE)
        // Column type is the 3rd field in the table
        ADO_VARIABLE_LENGTH_ENTRY2(3,adVarChar,m_szt_Type,
            sizeof(m_szt_Type),lt_TypeStatus,TRUE)
    END_ADO_BINDING()

public:
    CHAR m_szt_Title_id[7];
    ULONG lt_Title_idStatus;
    CHAR m_szt_Type[13];
    ULONG lt_TypeStatus;
};
// EndStatusH

```

See Also

[Status Property \(ADO Recordset\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 


```

//                                                                    //
//          StayInSyncX Function                                       //
//                                                                    //
//                                                                    //
//                                                                    //
void StayInSyncX(void)
{
    HRESULT hr = S_OK;

    // Define string variables.
    _bstr_t strCnn("Provider='MSDataShape';"
        "Data Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI';");

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _ConnectionPtr pConnection      = NULL;
    _RecordsetPtr  pRst             = NULL;
    _RecordsetPtr  pRstTitleAuthor = NULL;

    try
    {
        TESTHR(pRstTitleAuthor.CreateInstance(__uuidof(Recordset)));

        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));

        TESTHR(pRst.CreateInstance(__uuidof(Recordset)));

        // Open connection.
        pConnection->Open (strCnn, "", "", adConnectUnspecified);
        pRst->PutStayInSync(true);

        // Open recordset with names from Author & titleauthor table
        pRst->Open("SHAPE {select * from authors} "
            "APPEND ({select * from titleauthor}"
            "RELATE au_id TO au_id) AS chapTitleAuthor",
            _variant_t((IDispatch*)pConnection,true),
            adOpenStatic, adLockReadOnly, adCmdText);

        pRstTitleAuthor = pRst->GetFields()->GetItem("chapTitleAutho
            Value;
        int intLineCnt=0;
        while(!(pRst->EndOfFile))
        {
            printf("\n%s %s %s %s\n", (LPCSTR)(_bstr_t)pRst->
                Fields->GetItem("au_fname")->Value,
                (LPCSTR)(_bstr_t)pRst->Fields->GetItem("au_lname")->v
                (LPCSTR)(_bstr_t)pRst->Fields->GetItem("state")->Valu
                (LPCSTR)(_bstr_t)pRst->Fields->GetItem("au_id")->Valu

```

```

        intLineCnt++;

        if (intLineCnt%15 == 0)
        {
            printf("\nPress any key to continue...\n");
            getch();
        }

        _variant_t vIndex;
        while(!(pRstTitleAuthor->EndOfFile))
        {
            vIndex = (short) 0;
            printf("%s      ",(LPCSTR)(_bstr_t)pRstTitleAuthor->
                Fields->Item[&vIndex]->Value);
            vIndex = (short) 1;
            printf("%s      ",(LPCSTR)(_bstr_t)pRstTitleAuthor->
                Fields->Item[&vIndex]->Value);
            vIndex = (short) 2;
            printf("%s      ",(LPCSTR)(_bstr_t)pRstTitleAuthor->
                Fields->Item[&vIndex]->Value);
            vIndex = (short) 3;
            printf("%s\n", (LPCSTR)(_bstr_t)pRstTitleAuthor->
                Fields->Item[&vIndex]->Value);
            intLineCnt++;

            if (intLineCnt%15 == 0)
            {
                printf("\nPress any key to continue...\n");
                getch();
            }
            pRstTitleAuthor->MoveNext();
        }
        pRst->MoveNext();
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
if (pRst)
    if (pRst->State == adStateOpen)
        pRst->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)

```

```

        pConnection->Close();
    }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                             //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndStayInSyncCpp

```

See Also

[Fields Collection](#) | [Recordset Object](#) | [StayInSync Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Supports Method Example (VC++)

This example uses the [Supports](#) method to display the options supported by a recordset opened with different [cursor](#) types. The DisplaySupport function is required for this example to run.

```
// BeginSupportsCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <conio.h>

//Function Declarations.
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void SupportsX(void);
void DisplaySupport(_RecordsetPtr pRstTemp);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//    Main Function                                                    //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    SupportsX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//    SupportsX Function                                              //
//                                                                    //
////////////////////////////////////
void SupportsX(void)
{
    // Define ADO object pointers.
    // Initialize pointers on define.
```

```

// These are in the ADODB:: namespace.
_RecordsetPtr pRstTitles = NULL;

// Define Other Variables
HRESULT hr = S_OK;

// Assign connection string to a variable
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

try
{
    // Open a recordset from Titles table
    TESTHR(pRstTitles.CreateInstance(__uuidof(Recordset)));

    // Fill array with CursorType constants.
    int aintCursorType[4];
    aintCursorType[0] = adOpenForwardOnly;
    aintCursorType[1] = adOpenKeyset;
    aintCursorType[2] = adOpenDynamic;
    aintCursorType[3] = adOpenStatic;

    // Open recordset using each CursorType and optimistic locki
    // Then call the DisplaySupport procedure to display the
    // supported options.
    for (int intIndex=0; intIndex <= 3; intIndex++)
    {
        pRstTitles->CursorType =
            (enum CursorTypeEnum)aintCursorType[intIndex];
        pRstTitles->LockType = adLockOptimistic;

        // Pass the Cursor type and LockType to the Recordset.
        pRstTitles->Open ("titles", strCnn,
            (enum CursorTypeEnum)aintCursorType[intIndex],
            adLockOptimistic, adCmdTable);

        switch(aintCursorType[intIndex])
        {
            case adOpenForwardOnly:
                printf("\nForwardOnly cursor supports:\n");
                break;

            case adOpenKeyset:
                printf("\nKeyset cursor supports:\n");
                break;

            case adOpenDynamic:
                printf("\nDynamic cursor supports:\n");
                break;
        }
    }
}

```



```

////////////////////////////////////
void DisplaySupport (_RecordsetPtr pRstTemp)
{
    // Fill array with cursor option constants.
    long  alngConstants[11];
    alngConstants[0] = adAddNew;
    alngConstants[1] = adApproxPosition;
    alngConstants[2] = adBookmark;
    alngConstants[3] = adDelete;
    alngConstants[4] = adFind;
    alngConstants[5] = adHoldRecords;
    alngConstants[6] = adMovePrevious;
    alngConstants[7] = adNotify;
    alngConstants[8] = adResync;
    alngConstants[9] = adUpdate;
    alngConstants[10] = adUpdateBatch;

    for(int intIndex=0; intIndex <= 10; intIndex++)
    {
        bool booSupports = pRstTemp->
            Supports( (enum CursorOptionEnum)alngConstants[intIndex]

        if(booSupports)
        {
            switch(alngConstants[intIndex])
            {
                case adAddNew :
                    printf("\n  AddNew");
                    break;

                case adApproxPosition :
                    printf("\n  AbsolutePosition and AbsolutePage");
                    break;

                case adBookmark :
                    printf("\n  Bookmark");
                    break;

                case adDelete :
                    printf("\n  Delete");
                    break;

                case adFind :
                    printf("\n  Find");
                    break;

                case adHoldRecords :
                    printf("\n  Holding Records");
                    break;
            }
        }
    }
}

```

```

        case adMovePrevious :
            printf("\n MovePrevious and Move");
            break;

        case adNotify :
            printf("\n Notifications");
            break;

        case adResync :
            printf("\n Resyncing data");
            break;

        case adUpdate :
            printf("\n Update");
            break;

        case adUpdateBatch :
            printf("\n Batch updating");
            break;

        default :
            break;
    }
}

////////////////////////////////////
//                                                                    //
//    PrintProviderError Function                                     //
//                                                                    //
////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

```

```

}

////////////////////////////////////
//                                                                    //
//    PrintComError Function                                          //
//                                                                    //
////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndSupportsCpp

```

See Also

[Recordset Object](#) | [Supports Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Type Property Example (Field) (VC++)

This example demonstrates the [Type](#) property by displaying the name of the constant that corresponds to the value of the **Type** property of all the [Field](#) objects in the *Employees* table. The FieldType function is required for this procedure to run.

```
// BeginTypeFieldCpp
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void TypeX(void);
_bstr_t FieldType(int intType);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//    Main Function                                                    //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    TypeX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//                TypeX Function                                       //
//                                                                    //
////////////////////////////////////
```

```

void TypeX(void)
{
    HRESULT hr = S_OK;

    // Define string variables.
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr pRstEmployees = NULL;
    FieldsPtr pFldLoop = NULL;

    try
    {
        // Open recordset with data from Employee table.
        TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));
        pRstEmployees->Open ("employee", strCnn,
            adOpenForwardOnly, adLockReadOnly, adCmdTable);

        printf("Fields in Employee Table:\n\n");

        // Enumerate the Fields collection of the Employees table.
        pFldLoop = pRstEmployees->GetFields();
        int intLine = 0;
        for (int intFields = 0; intFields < (int)pFldLoop->
            GetCount(); intFields++)
        {
            _variant_t Index;
            Index.vt = VT_I2;
            Index.iVal = intFields;
            printf(" Name: %s\n" ,
                (LPCSTR) pFldLoop->GetItem(Index)->GetName());
            printf(" Type: %s\n\n",
                (LPCTSTR)FieldType(pFldLoop->GetItem(Index)->Type));
            intLine++;
            if(intLine % 5 == 0)
            {
                printf("Press any key to continue...");
                getch();
                system("cls");
            }
        }
    }
    catch (_com_error &e)
    {
        // Notify the user of errors if any.
        // Pass a connection pointer accessed from the Recordset.
        _variant_t vtConnect = pRstEmployees->GetActiveConnection();
    }
}

```

```

// GetActiveConnection returns connect string if connection
// is not open, else returns Connection object.
switch(vtConnect.vt)
{
    case VT_BSTR:
        PrintComError(e);
        break;
    case VT_DISPATCH:
        PrintProviderError(vtConnect);
        break;
    default:
        printf("Errors occured.");
        break;
}
}

// Clean up objects before exit.
if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
}

////////////////////////////////////
//                                     //
//          FieldType Function          //
//                                     //
////////////////////////////////////
_bstr_t FieldType(int intType)
{
    _bstr_t strType;
    switch(intType)
    {
        case adChar:
            strType = "adChar";
            break;
        case adVarChar:
            strType = "adVarChar";
            break;
        case adSmallInt:
            strType = "adSmallInt";
            break;
        case adUnsignedTinyInt:
            strType = "adUnsignedTinyInt";
            break;
        case adDBTimeStamp:
            strType = "adDBTimeStamp";
            break;
        default:

```

```

        break;
    }
    return strType;
}

////////////////////////////////////
//                                                                    //
//    PrintProviderError Function                                     //
//                                                                    //
////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////
//                                                                    //
//    PrintComError Function                                       //
//                                                                    //
////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndTypeFieldCpp

```

See Also

[Field Object](#) | [Type Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Type Property Example (Property) (VC++)

This example demonstrates the [Type](#) property. It is a model of a utility for listing the names and types of a collection, like [Properties](#), [Fields](#), etc.

We do not need to open the [Recordset](#) to access its **Properties** collection; they come into existence when the **Recordset** object is instantiated. However, setting the [CursorLocation](#) property to **adUseClient** adds several [dynamic properties](#) to the **Recordset** object's **Properties** collection, making the example a little more interesting. For sake of illustration, we explicitly use the [Item](#) property to access each [Property](#) object.

```
// BeginTypePropertyCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include<conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void TypeX();
void PrintComError(_com_error &e);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    TypeX();

    ::CoUninitialize();
}

////////////////////////////////////
```

```

//                                                    //
//          TypeX Function                               //
//                                                    //
////////////////////////////////////
void TypeX()
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace
    _RecordsetPtr pRst = NULL;
    PropertyPtr pProperty = NULL;

    //Define Other Variables
    _bstr_t strMsg;
    _variant_t vIndex;
    int intLineCnt = 0;

    try
    {
        TESTHR(pRst.CreateInstance (__uuidof(Recordset)));

        // Set the Recordset Cursor Location
        pRst->CursorLocation = adUseClient;

        for (short iIndex = 0; iIndex <= (pRst->Properties->
            GetCount() - 1);iIndex++)
        {
            vIndex = iIndex;
            pProperty = pRst->Properties->GetItem(vIndex);

            int propType = (int)pProperty->GetType();
            switch(propType)
            {
                case adBigInt:
                    strMsg = "adBigInt";
                    break;
                case adBinary:
                    strMsg = "adBinary";
                    break;
                case adBoolean:
                    strMsg = "adBoolean";
                    break;
                case adBSTR:
                    strMsg = "adBSTR";
                    break;
                case adChapter:
                    strMsg = "adChapter";
                    break;
            }
        }
    }
}

```

```
case adChar:
    strMsg = "adChar";
    break;
case adCurrency:
    strMsg = "adCurrency";
    break;
case adDate:
    strMsg = "adDate";
    break;
case adDBDate:
    strMsg = "adDBDate";
    break;
case adDBTime:
    strMsg = "adDBTime";
    break;
case adDBTimeStamp:
    strMsg = "adDBTimeStamp";
    break;
case adDecimal:
    strMsg = "adDecimal";
    break;
case adDouble:
    strMsg = "adDouble";
    break;
case adEmpty:
    strMsg = "adEmpty";
    break;
case adError:
    strMsg = "adError";
    break;
case adFileTime:
    strMsg = "adFileTime";
    break;
case adGUID:
    strMsg = "adGUID";
    break;
case adIDispatch:
    strMsg = "adIDispatch";
    break;
case adInteger:
    strMsg = "adInteger";
    break;
case adIUnknown:
    strMsg = "adIUnknown";
    break;
case adLongVarBinary:
    strMsg = "adLongVarBinary";
    break;
case adLongVarChar:
```

```
        strMsg = "adLongVarChar";
        break;
case adLongVarWChar:
    strMsg = "adLongVarWChar";
    break;
case adNumeric:
    strMsg = "adNumeric";
    break;
case adPropVariant:
    strMsg = "adPropVariant";
    break;
case adSingle:
    strMsg = "adSingle";
    break;
case adSmallInt:
    strMsg = "adSmallInt";
    break;
case adTinyInt:
    strMsg = "adTinyInt";
    break;
case adUnsignedBigInt:
    strMsg = "adUnsignedBigInt";
    break;
case adUnsignedInt:
    strMsg = "adUnsignedInt";
    break;
case adUnsignedSmallInt:
    strMsg = "adUnsignedSmallInt";
    break;
case adUnsignedTinyInt:
    strMsg = "adUnsignedTinyInt";
    break;
case adUserDefined:
    strMsg = "adUserDefined";
    break;
case adVarBinary:
    strMsg = "adVarBinary";
    break;
case adVarChar:
    strMsg = "adVarChar";
    break;
case adVariant:
    strMsg = "adVariant";
    break;
case adVarNumeric:
    strMsg = "adVarNumeric";
    break;
case adVarWChar:
    strMsg = "adVarWChar";
    break;
```

```

        case adWChar:
            strMsg = "adWChar";
            break;
        default:
            strMsg = "*UNKNOWN*";
            break;
    }

    intLineCnt++;
    if (intLineCnt%20 == 0)
    {
        printf("\nPress any key to continue...\n");
        getch();
    }
    printf ("Property %d : %s,Type = %s\n",iIndex,
        (LPCSTR)pProperty->GetName(),(LPCSTR)strMsg);
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    PrintComError(e);
}
}

////////////////////////////////////
//                                                                    //
//      PrintComError Function                                         //
//                                                                    //
////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndTypePropertyCpp

```

See Also

Property Object | Type Property

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Update and CancelUpdate Methods Example (VC++)

This example demonstrates the [Update](#) method in conjunction with the [CancelUpdate](#) method.

```
// BeginUpdateCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <stdio.h>
#include <ole2.h>
#include <malloc.h>
#include <conio.h>
#include "UpdateX.h"

// Function Declartion.
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void UpdateX(void);
void UpdateX2(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    UpdateX();

    //Wait here for user to see the output..
    printf("\nPress any key to continue...");
    getch();

    //Clear the screen for the next display
    system("cls");

    UpdateX2();

    ::CoUninitialize();
}

////////////////////////////////////
//
```

```

//      UpdateX Function                                     //
//                                                                 //
//////////////////////////////////////////////////////////////////
void UpdateX(void)
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _RecordsetPtr pRstEmployees = NULL;

    // Define string variables.
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    IADORecordBinding *picRs = NULL; // Interface Pointer declara
    CEmployeeRs emprs; // C++ Class object.

    try
    {
        // Open recordset with names from Employee table.
        TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));
        pRstEmployees->CursorType = adOpenKeyset;
        pRstEmployees->LockType = adLockOptimistic;
        pRstEmployees->Open("SELECT fname, lname FROM Employee "
            "ORDER BY lname",strCnn,adOpenKeyset,adLockOptimistic,
            adCmdText);

        // Store original data.
        _bstr_t strOldFirst = pRstEmployees->Fields->
            GetItem("fname")->Value;
        _bstr_t strOldLast = pRstEmployees->Fields->
            GetItem("lname")->Value;

        //Change data in edit buffer.
        pRstEmployees->Fields->GetItem("fname")->Value =
            (_bstr_t)("Linda");
        pRstEmployees->Fields->GetItem("lname")->Value =
            (_bstr_t)("Kobara");

        // Show contents of buffer and get user input.
        printf("\n\nEdit in progress:\n\n");

        printf("Original data = %s %s \n",
            (LPSTR)strOldFirst,(LPSTR)strOldLast);

        printf("Data in buffer = %s %s",
            (LPSTR)(_bstr_t) pRstEmployees->Fields->
            GetItem("fname")->Value,\

```

```

        (LPSTR) (_bstr_t) pRstEmployees->Fields->
        GetItem("lname")->Value);

// Ask if the User wants to Update
printf("\n\nUse Update to replace the original data with the
      " buffered data in the Recordset? (y/n): ");
char chKey = getch();

if(toupper(chKey) == 'Y')
    pRstEmployees->Update();
else
    pRstEmployees->CancelUpdate();

//Open an IADORecordBinding interface pointer which
//we'll use for binding Recordset to a class.
TESTHR(pRstEmployees->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

//Bind the Recordset to a C++ Class here.
TESTHR(picRs->BindToRecordset(&emprs));

pRstEmployees->MoveFirst();

// Show the resulting data.
printf("\nData in recordset = %s %s", emprs.le_fnameStatus
      adFldOK ? emprs.m_size_fname : "<NULL>",
      emprs.le_lnameStatus == adFldOK ?
      emprs.m_size_lname : "<NULL>");

// Restore original data because this is a demonstration.
if ((strcmp((char *)strOldFirst, emprs.m_size_fname) &&
    strcmp((char *)strOldLast, emprs.m_size_lname)))
{
    pRstEmployees->Fields->GetItem("fname")->Value = strOldF
    pRstEmployees->Fields->GetItem("lname")->Value = strOldL
    pRstEmployees->Update();
}
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstEmployees->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:

```

```

        PrintComError(e);
        break;
    case VT_DISPATCH:
        PrintProviderError(vtConnect);
        break;
    default:
        printf("Errors occurred.");
        break;
    }
}

// Clean up objects before exit.
//Release the IADORecordset Interface here
if (picRs)
    picRs->Release();

if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
}

// The next example demonstrates the Update method
// in conjunction with the AddNew method.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//          UpdateX2 Function                                          //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void UpdateX2(void)
{
    HRESULT hr = S_OK;

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADODB:: namespace.
    _ConnectionPtr pConnection      = NULL;
    _RecordsetPtr  pRstEmployees = NULL;

    // Define string variables.
    _bstr_t strCnn("Provider='sqloledb';Data Source='MySQLServer';"
        "Initial Catalog='pubs';Integrated Security='SSPI'");

    IADORecordBinding *picRs = NULL; // Interface Pointer declar
    CEmployeeRs1 emprs;           // C++ Class object.

    try
    {
        // Open a connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));

```

```

pConnection->Open(strCnn, "", "", NULL);

// Open recordset with data from Employee table.
TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));
pRstEmployees->CursorType = adOpenKeyset;
pRstEmployees->LockType = adLockOptimistic;
pRstEmployees->Open("employee",
    _variant_t((IDispatch*)pConnection, true),
    adOpenKeyset, adLockOptimistic, adCmdTable);

pRstEmployees->AddNew();
_bstr_t strEmpID = "B-S55555M";
pRstEmployees->Fields->GetItem("emp_id")->Value = strEmpID;
pRstEmployees->Fields->GetItem("fname")->Value =
    (_bstr_t) ("Bill");
pRstEmployees->Fields->GetItem("lname")->Value =
    (_bstr_t) ("Sornsin");

// Show contents of buffer and get user input.
printf("\n\nAddNew in progress:\n\n");

printf("Data in buffer = %s , %s %s",
    (LPSTR) (_bstr_t) pRstEmployees->Fields->
    GetItem("emp_id")->Value,
    (LPSTR) (_bstr_t) pRstEmployees->Fields->
    GetItem("fname")->Value,
    (LPSTR) (_bstr_t) pRstEmployees->Fields->
    GetItem("lname")->Value);

printf("\n\nUse Update to save buffer to recordset?(y/n):");
char chKey = getch();

if(toupper(chKey) == 'Y')
{
    pRstEmployees->Update();

    //Open an IADORecordBinding interface pointer which
    //we'll use for binding Recordset to a class.
    TESTHR(pRstEmployees->QueryInterface(
        __uuidof(IADORecordBinding), (LPVOID*)&picRs));

    //Bind the Recordset to a C++ Class here
    TESTHR(picRs->BindToRecordset(&emprs));

    // Go to the new record and show the resulting data.
    printf ("\n\nData in recordset = %s , %s %s",
        emprs.le_empidStatus == adFldOK ?
        emprs.m_size_empid : "<NULL>",
        emprs.le_fnameStatus == adFldOK ?

```

```

        emprs.m_size_fname : "<NULL>",
        emprs.le_lnameStatus == adFldOK ?
        emprs.m_size_lname : "<NULL>");
    }
    else
    {
        pRstEmployees->CancelUpdate();
        printf("\n\nNo new record added.\n");
    }
    // Delete new data because this is a demonstration.
    _bstr_t strSQLDelete("DELETE FROM employee WHERE emp_id = '"
        strEmpID + "'");
    pConnection->Execute(strSQLDelete , NULL, adExecuteNoRecords);
}

catch(_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Connection.
    PrintProviderError(pConnection);
    PrintComError(e);
}

// Clean up objects before exit.
//Release the IADORRecordset Interface here
if (picRs)
    picRs->Release();

if (pRstEmployees)
    if (pRstEmployees->State == adStateOpen)
        pRstEmployees->Close();
if (pConnection)
    if (pConnection->State == adStateOpen)
        pConnection->Close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      PrintProviderError Function                                     //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;

```

```

        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                 //
//      PrintComError Function                                     //
//                                                                 //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndUpdateCpp

```

UpdateX.h

```

// BeginUpdateH
#include "icrsint.h"

//This Class extracts only fname,lname from employee table.
class CEmployeeRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)
    // fname is the 1st field in the recordset
    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_size_fname,
        sizeof(m_size_fname), le_fnameStatus, FALSE)
    // lname is the 2nd field in the recordset.
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_size_lname,
        sizeof(m_size_lname), le_lnameStatus, FALSE)

END_ADO_BINDING()

public:

```

```

    CHAR    m_size_lname[31];
    ULONG   le_lnameStatus;
    CHAR    m_size_fname[21];
    ULONG   le_fnameStatus;
};

//This Class extracts only empid,fname,lname,from employee table.
class CEmployeeRs1 : public CADORRecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs1)
    // emp_id is the 1st field in the table.
    ADO_VARIABLE_LENGTH_ENTRY2(1, adVarChar, m_size_empid,
        sizeof(m_size_empid), le_empidStatus, FALSE)
    // fname is the 2nd field in the table.
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_size_fname,
        sizeof(m_size_fname), le_fnameStatus, FALSE)
    // lname is the 4rt field in the table.
    ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_size_lname,
        sizeof(m_size_lname), le_lnameStatus, FALSE)

END_ADO_BINDING()

public:
    CHAR    m_size_empid[10];
    ULONG   le_empidStatus;
    CHAR    m_size_lname[31];
    ULONG   le_lnameStatus;
    CHAR    m_size_fname[21];
    ULONG   le_fnameStatus;
};
// EndUpdateH

```

See Also

[CancelUpdate Method](#) | [Update Method](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

UpdateBatch and CancelBatch Methods Example (VC++)

This example demonstrates the [UpdateBatch](#) method in conjunction with the [CancelBatch](#) method.

```
// BeginUpdateBatchCpp
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>
#include "UpdateBatchX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void UpdateBatchX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                                     //
//      Main Function                   //
//                                     //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    UpdateBatchX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                     //
//      UpdateBatchX Function           //
//                                     //
////////////////////////////////////
void UpdateBatchX(void)
{
    HRESULT hr = S_OK;
```

```

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADO:: namespace.
_RecordsetPtr pRstTitles = NULL;

// Define string variables.
_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI'");

IADORecordBinding *picRs = NULL; // Interface Pointer Decl
CTitleRs titlers; // C++ Class Object

try
{
    // Open titles table.
    TESTHR(pRstTitles.CreateInstance(__uuidof(Recordset)));
    pRstTitles->CursorType = adOpenKeyset;
    pRstTitles->LockType = adLockBatchOptimistic;
    pRstTitles->Open ("titles", strCnn,
                    adOpenKeyset, adLockBatchOptimistic, adCmdTable);

    // Open IADORecordBinding interface pointer for binding
    // Recordset to a class
    TESTHR(pRstTitles->QueryInterface(
        __uuidof(IADORecordBinding), (LPVOID*)&picRs));

    // Binding the Recordset to a C++ Class
    TESTHR(picRs->BindToRecordset(&titlers));
    pRstTitles->MoveFirst();

    // Loop through recordset and ask user if she wants,
    // to change the type for a specified title.
    while (!(pRstTitles->EndOfFile))
    {
        // Compare type with psychology
        if (!strcmp( (char *)strtok(titlers.m_szt_Type, " "),
                    "psychology" ))
        {
            printf("\n\nTitle: %s \nChange type to self_help?(y/
                    titlers.m_szt_Title);
            char chKey;
            chKey = getch();
            if(toupper(chKey) == 'Y')
            {
                // Change type to self_help.
                pRstTitles->Fields->GetItem("type")->Value =
                    (_bstr_t)("self_help");
            }
        }
    }
}

```

```

    pRstTitles->MoveNext();
}

// Ask the user if she wants to commit to all the
// changes made above.
printf("\n\nSave all changes?");
char chKey;
chKey = getch();
if(toupper(chKey) == 'Y')
{
    pRstTitles->UpdateBatch(adAffectAll);
}
else
{
    pRstTitles->CancelBatch(adAffectAll);
}

// Print current data in recordset.
pRstTitles->Requery(adOptionUnspecified);

// Open IADORecordBinding interface pointer for Binding Reco
TESTHR(pRstTitles->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

// ReBinding the Recordset to a C++ Class.
TESTHR(picRs->BindToRecordset(&titlers));

// Move to the first record of the title table
pRstTitles->MoveFirst();

//Clear the screen for the next display.
system("cls");

while (!pRstTitles->EndOfFile)
{
    printf("%s - %s\n",
        titlers.lt_TitleStatus == adFldOK ?
        titlers.m_szt_Title : "<NULL>",
        titlers.lt_TypeStatus == adFldOK ?
        titlers.m_szt_Type : "<NULL>");
    pRstTitles->MoveNext();
}

pRstTitles->MoveFirst();

// Restore original data because this is demonstration.
while (!pRstTitles->EndOfFile)
{
    // Compare type with psychology

```

```

        if(!strcmp( (char *)strtok(titlers.m_szt_Type, " "),
            "self_help" ))
        {
            // Change type to psychology.
            pRstTitles->Fields->GetItem("type")->Value =
                (_bstr_t)("psychology");
        }
        pRstTitles->MoveNext();
    }
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.
    _variant_t vtConnect = pRstTitles->GetActiveConnection();

    // GetActiveConnection returns connect string if connection
    // is not open, else returns Connection object.
    switch(vtConnect.vt)
    {
        case VT_BSTR:
            PrintComError(e);
            break;
        case VT_DISPATCH:
            PrintProviderError(vtConnect);
            break;
        default:
            printf("Errors occured.");
            break;
    }
}

if (pRstTitles)
    if (pRstTitles->State == adStateOpen)
    {
        pRstTitles->UpdateBatch(adAffectAll);
        pRstTitles->Close();
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//    PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

```

```

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndUpdateBatchCpp

```

UpdateBatchX.h

```

// BeginUpdateBatchH
#include "icrsint.h"

//This class extracts titles and type from Titles table
class CTitlers : public CADORecordBinding
{
BEGIN_ADO_BINDING(CTitlers)
    // Column title is the 2nd field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(2,adVarChar,m_szt_Title,
        sizeof(m_szt_Title),lt_TitleStatus,FALSE)
    // Column type is the 3rd field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(3,adVarChar,m_szt_Type,
        sizeof(m_szt_Type),lt_TypeStatus,TRUE)

```

```
END_ADO_BINDING()  
  
public:  
    CHAR m_szt_Title[81];  
    ULONG lt_TitleStatus;  
    CHAR m_szt_Type[13];  
    ULONG lt_TypeStatus;  
};  
// EndUpdateBatchH
```

See Also

[CancelBatch Method](#) | [UpdateBatch Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Value Property Example (VC++)

This example demonstrates the [Value](#) property with [Field](#) and [Property](#) objects by displaying field and property values for the *Employees* table.

```
// BeginValueCpp
#import "c:\Program Files\Common Files\System\AD0\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void ValueX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

/////////////////////////////////////////////////////////////////
//                                                                    //
//    Main Function                                                    //
//                                                                    //
/////////////////////////////////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    ValueX();

    ::CoUninitialize();
}

/////////////////////////////////////////////////////////////////
//                                                                    //
//    ValueX Function                                                  //
//                                                                    //
/////////////////////////////////////////////////////////////////
void ValueX(void)
{
    HRESULT hr = S_OK;

    // Define string variables.
```

```

_bstr_t strCnn("Provider='sqloledb';Data Source='MySqlServer';"
              "Initial Catalog='pubs';Integrated Security='SSPI';");

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
_RecordsetPtr pRstEmployees = NULL;
FieldsPtr     pFldLoop      = NULL;
PropertiesPtr pPrpLoop      = NULL;
_variant_t vtIndex;
vtIndex.vt = VT_I2;

try
{
    // Open recordset with data from Employee table.
    TESTHR(pRstEmployees.CreateInstance(__uuidof(Recordset)));
    pRstEmployees->Open ("employee",strCnn ,
                       adOpenForwardOnly, adLockReadOnly, adCmdTable);

    printf("Field values in rstEmployees\n\n");

    // Enumerate the Fields collection of the Employees table.
    pFldLoop = pRstEmployees->GetFields();

    for (int intFields = 0; intFields < (int)pFldLoop->GetCount(
    {
        vtIndex.iVal = intFields;

        // Because Value is the default property of a
        // Field object,the use of the actual keyword
        // here is optional.
        printf(" %s = %s\n\n" ,
              (LPCSTR) pFldLoop->GetItem(vtIndex)->GetName(),
              (LPCSTR) (_bstr_t) pFldLoop->GetItem(vtIndex)->Value
        }

    printf("Press any key to continue...\n\n");
    getch();
    printf("Property values in rstEmployees\n\n");

    // Enumerate the Properties collection of the Recordset obje
    pPrpLoop = pRstEmployees->GetProperties();
    int intLine = 0;

    for (int intProperties = 0; intProperties < (int)pPrpLoop->
        GetCount(); intProperties++)
    {
        vtIndex.iVal = intProperties;

        // Because Value is the default property of a

```

```

// Property object, the use of the actual keyword
// here is optional.
_variant_t propValue = pPrpLoop->GetItem(vtIndex)->Value
switch(propValue.vt)
{
case (VT_BOOL):
    if(propValue.boolVal)
    {
        printf(" %s = True\n\n", (LPCSTR) pPrpLoop->
            GetItem(vtIndex)->GetName());
    }
    else
    {
        printf(" %s = False\n\n", (LPCSTR) pPrpLoop->
            GetItem(vtIndex)->GetName());
    }
    break;

case (VT_I4):
    printf(" %s = %d\n\n", (LPCSTR) pPrpLoop->
        GetItem(vtIndex)->GetName(),
        pPrpLoop->GetItem(vtIndex)->Value.lVal);
    break;

case (VT_EMPTY):
    printf(" %s = \n\n", (LPCSTR) pPrpLoop->
        GetItem(vtIndex)->GetName());
    break;

default:
    break;
}

intLine++;
if (intLine % 10 == 0)
{
    printf("\nPress any key to continue...");
    getch();

    //Clear the screen for the next display
    system("cls");
}
}
}
catch (_com_error &e)
{
    // Notify the user of errors if any.
    // Pass a connection pointer accessed from the Recordset.

```



```
//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndValueCpp
```

See Also

[Field Object](#) | [Property Object](#) | [Value Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Version Property Example (VC++)

This example uses the [Version](#) property of a [Connection](#) object to display the current ADO version. It also uses several [dynamic properties](#) to show:

- the current DBMS name and version.
- OLE DB version.
- [provider](#) name and version.
- [ODBC](#) version.
- ODBC driver name and version.

```
// BeginVersionCpp
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

#include <ole2.h>
#include <stdio.h>
#include <conio.h>

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void VersionX(void);
void PrintProviderError(_ConnectionPtr pConnection);
void PrintComError(_com_error &e);

////////////////////////////////////
//                               //
//      Main Function             //
//                               //
////////////////////////////////////

void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    VersionX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//      VersionX Function        //
//                               //
////////////////////////////////////
```

```

//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void VersionX(void)
{
    HRESULT    hr = S_OK;

    // Define string variables.
    _bstr_t strCnn("driver={SQL Server};server='MySqlServer';"
        "user id='MyUserId';password='MyPassword';database='pubs';")

    // Define ADO object pointers.
    // Initialize pointers on define.
    // These are in the ADO:: namespace.
    _ConnectionPtr    pConnection    = NULL;

    try
    {
        // Open connection.
        TESTHR(pConnection.CreateInstance(__uuidof(Connection));
        pConnection->Open (strCnn, "", "", adConnectUnspecified);

        printf("ADO Version    : %s\n\n", (LPCSTR) pConnection->Version);
        printf("DBMS Name     : %s\n\n", (LPCSTR) (_bstr_t)
            pConnection->Properties->GetItem("DBMS Name")->Value);
        printf("DBMS Version    : %s\n\n", (LPCSTR) (_bstr_t)
            pConnection->Properties->GetItem("DBMS Version")->Value);
        printf("OLE DB Version   : %s\n\n", (LPCSTR) (_bstr_t)
            pConnection->Properties->GetItem("OLE DB Version")->Value);
        printf("Provider Name    : %s\n\n", (LPCSTR) (_bstr_t)
            pConnection->Properties->GetItem("Provider Name")->Value);
        printf("Provider Version  : %s\n\n", (LPCSTR) (_bstr_t)
            pConnection->Properties->GetItem("Provider Version")->Value);
        printf("Driver Name     : %s\n\n", (LPCSTR) (_bstr_t)
            pConnection->Properties->GetItem("Driver Name")->Value);
        printf("Driver Version    : %s\n\n", (LPCSTR) (_bstr_t)
            pConnection->Properties->GetItem("Driver Version")->Value);
        printf("Driver ODBC Version : %s\n\n", (LPCSTR) (_bstr_t)
            pConnection->Properties->GetItem("Driver ODBC Version")->Value);

    }

    catch (_com_error &e)
    {
        // Notify the user of errors if any.
        PrintProviderError(pConnection);
        PrintComError(e);
    }

    if (pConnection)
        if (pConnection->State == adStateOpen)

```

```

        pConnection->Close();
    }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintProviderError Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
    // Print Provider Errors from Connection object.
    // pErr is a record object in the Connection's Error collection.
    ErrorPtr    pErr = NULL;

    if( (pConnection->Errors->Count) > 0)
    {
        long nCount = pConnection->Errors->Count;
        // Collection ranges from 0 to nCount -1.
        for(long i = 0; i < nCount; i++)
        {
            pErr = pConnection->Errors->GetItem(i);
            printf("Error number: %x\t%s\n", pErr->Number,
                (LPCSTR) pErr->Description);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      PrintComError Function                                                                     //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void PrintComError(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    // Print Com errors.
    printf("Error\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s\n", e.ErrorMessage());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndVersionCpp

```

See Also

[Connection Object](#) | [Version Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ADO Code Examples in Microsoft Visual J++

Use the following code examples to learn how to use the ADO methods and properties when writing in Microsoft Visual J++.

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

Methods

- [AddNew Method Example](#)
- [Append and CreateParameter Methods Example](#)
- [AppendChunk and GetChunk Methods Example](#)
- [BeginTrans, CommitTrans, and RollbackTrans Methods Example](#)
- [Cancel Method Example](#)
- [Clone Method Example](#)
- [CompareBookmarks Method Example](#)
- [Delete Method Example](#)
- [Execute, Requery, and Clear Methods Example](#)
- [Find Method Example](#)
- [GetRows Method Example](#)
- [GetString Method Example](#)
- [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods Example](#)
- [NextRecordset Method Example](#)
- [Open and Close Methods Example](#)
- [OpenSchema Method Example](#)
- [Refresh Method Example](#)
- [Resync Method Example](#)
- [Save and Open Methods Example](#)
- [Supports Method Example](#)
- [Update and CancelUpdate Methods Example](#)
- [UpdateBatch and CancelBatch Methods Example](#)

Properties

- [AbsolutePage, PageCount, and PageSize Properties Example](#)
- [AbsolutePosition and CursorLocation Properties Example](#)
- [ActiveCommand Property Example](#)
- [ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example](#)
- [ActualSize and DefinedSize Properties Example](#)
- [Attributes and Name Properties Example](#)
- [BOF, EOF, and Bookmark Properties Example](#)
- [CacheSize Property Example](#)
- [ConnectionString, ConnectionTimeout, and State Properties Example](#)
- [Count Property Example](#)
- [CursorType, LockType, and EditMode Properties Example](#)
- [Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState Properties Example](#)
- [Filter and RecordCount Properties Example](#)
- [IsolationLevel and Mode Properties Example](#)
- [Item Property Example](#)
- [MarshalOptions Property Example](#)
- [MaxRecords Property Example](#)
- [NumericScale and Precision Properties Example](#)
- [Optimize Property Example](#)
- [OriginalValue and UnderlyingValue Properties Example](#)
- [Prepared Property Example](#)
- [Provider and DefaultDatabase Properties Example](#)
- [Sort Property Example](#)
- [Source Property Example](#)
- [State Property Example](#)
- [Status Property Example](#)
- [StayInSync Property Example](#)
- [Type Property Example \(Fields\)](#)
- [Type Property Example \(Property\)](#)
- [Value Property Example](#)
- [Version Property Example](#)

See Also

[ADO Code Examples in Microsoft Visual Basic](#) | [ADO Code Examples in Microsoft Visual Basic Scripting Edition](#) | [ADO Code Examples in Microsoft](#)

Visual C++

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

AbsolutePage, PageCount, and PageSize Properties Example (VJ++)

This example uses the [AbsolutePage](#), [PageCount](#), and [PageSize](#) properties to display names and hire dates from the *Employees* table, five records at a time.

```
// BeginAbsolutePageJ
// The WFC class includes the ADO objects.

import com.ms.wfc.data.*;
import java.io.* ;

public class AbsolutePageX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        AbsolutePageX();
        System.exit(0);
    }

    // AbsolutePageX function

    static void AbsolutePageX()
    {
        // Define ADO Objects.
        Recordset rstEmployees = null;

        // Declarations.
        BufferedReader in = new BufferedReader (new
            InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        String strName;
        String strFName;
        String strLName;
        String strHDate;
        int intPage;
        int intRecord;

        try
```

```

{
    rstEmployees = new Recordset();

    // Use client cursor to enable AbsolutePosition property.
    rstEmployees.setCursorLocation( AdoEnums.CursorLocation.CLI

    // Open a recordset using client cursor for the Employees t
    rstEmployees.open("employee", strCnn,
        AdoEnums.CursorType.FORWARDONLY,
        AdoEnums.LockType.READONLY,
        AdoEnums.CommandType.TABLE);

    // Display names and hire dates, five records at a time.

    rstEmployees.setPageSize(5);
    int intPageCount = rstEmployees.getPageCount();
    for ( intPage = 1; intPage <= intPageCount; intPage++)
    {
        strName = "";
        rstEmployees.setAbsolutePage(intPage);
        for ( intRecord = 1; intRecord <= rstEmployees.getPageSi
            intRecord++)
        {
            strFName = rstEmployees.getField("fname").getString()
            strLName = rstEmployees.getField("lname").getString()
            strHDate = rstEmployees.getField("hire_date").getStri

            strHDate = strHDate.substring(5,7) + "/" +
                strHDate.substring(8,10) +
                "/" + strHDate.substring(2,4);

            strName = strName + "\n" + strFName + " " + strLName
                " " + strHDate;
            rstEmployees.moveNext();
            if ( rstEmployees.getEOF())
                break;
        }
        System.out.println(strName);
        // Get user input to display next records.

        System.out.println("\n\nPress <Enter> key to Continue.")
        line = in.readLine();
    }
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // Check for null pointer for connection object.
    if (rstEmployees.getActiveConnection()==null)

```

```

        System.out.println("Exception: " + ae.getMessage());

// As passing a Recordset, check for null pointer first.
if (rstEmployees != null)
{
    PrintProviderError(rstEmployees.getActiveConnection());
}
else
{
    System.out.println("Exception: " + ae.getMessage());
}
}
// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}
finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i < nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

```

```
}  
  
//.PrintIOError Function  
  
static void PrintIOError( java.io.IOException je)  
{  
    System.out.println("Error \n");  
    System.out.println("\tSource = " + je.getClass() + "\n");  
    System.out.println("\tDescription = " + je.getMessage() + "\n"  
}  
}  
// EndAbsolutePageJ
```

See Also

[AbsolutePage Property](#) | [PageCount Property](#) | [PageSize Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AbsolutePosition and CursorLocation Properties Example (VJ++)

This example demonstrates how the [AbsolutePosition](#) property can track the progress of a loop that enumerates all the records of a [Recordset](#). It uses the [CursorLocation](#) property to enable the **AbsolutePosition** property by setting the [cursor](#) to a [client](#) cursor.

```
// BeginAbsolutePositionJ
import com.ms.wfc.data.*;
import java.io.*;

public class AbsolutePositionX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        AbsolutePositionX();
        System.exit(0);
    }

    //.AbsolutePositionX function

    static void AbsolutePositionX()
    {

        // define ADO Objects.
        Recordset rstEmployees = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'";

        String strLName;
        String strMessage;
        String strAbsolutePosition, strRecordCount;
        int intAbsolutePosition;
        int intRecordCount;
        int intChoice;
```

```

try
{
    rstEmployees = new Recordset();

    // Use client cursor to enable AbsolutePosition property.
    rstEmployees.setCursorLocation( AdoEnums.CursorLocation.CLI

    // Open a recordset for Employees table using a client curs
    rstEmployees.open("employee", strCnn,
        AdoEnums.CursorType.FORWARDONLY ,
        AdoEnums.LockType.READONLY,
        AdoEnums.CommandType.TABLE);

    // Enumerate Recordset.
    while ( !rstEmployees.getEOF()) // continuous loop
    {
        intRecordCount = rstEmployees.getRecordCount();
        strRecordCount = Integer.toString(intRecordCount);

        // Read data field in the variables.
        strLName = rstEmployees.getField("lname").getString();
        intAbsolutePosition = rstEmployees.getAbsolutePosition()
        strAbsolutePosition = Integer.toString(intAbsolutePositi

        // Display current record information.
        strMessage = "\nEmployee: " + strLName + "\n" + "(Record
            strAbsolutePosition + " of " +strRecordCount + " )";
        System.out.println(strMessage);
        System.out.println(
            "\nDo you want to continue (1 -> Yes / 2 -> No)?");
        //user types a number followed by enter (cr-lf).
        line = in.readLine().trim();
        intChoice = Integer.parseInt(line);
        if ( intChoice != 1)
            break;
        rstEmployees.moveNext();
    }
}

catch( NumberFormatException ne)
{
    System.out.println("\nException : Integer Input required.")
    System.exit(0);
}

catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

```

```

// Check for null pointer for connection object.
if (rstEmployees.getActiveConnection()== null)
    System.out.println("Exception: " + ae.getMessage());
// As passing a Recordset, check for null pointer first.
if (rstEmployees != null)
{
    PrintProviderError(rstEmployees.getActiveConnection());
}
else
{
    System.out.println("Exception: " + ae.getMessage());
}
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
        }
    }
}

```

```
                System.out.println("\t Error number: " + ErrItem.getNumb
                    + "\t" + ErrItem.getDescription() );
            }
        }
    }

    //PrintIOError Function

    static void PrintIOError( java.io.IOException je)
    {
        System.out.println("Error \n");
        System.out.println("\tSource = " + je.getClass() + "\n");
        System.out.println("\tDescription = " + je.getMessage() + "\n"
    }
}

// EndAbsolutePositionJ
```

See Also

[AbsolutePosition Property](#) | [CursorLocation Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActiveCommand Property Example (VJ++)

This example demonstrates the [ActiveCommand](#) property.

A subroutine is given a [Recordset](#) object whose **ActiveCommand** property is used to display the command text and parameter that created the **Recordset**.

```
// BeginActiveCommandJ
import com.ms.wfc.data.*;
import java.io.* ;

public class ActiveCommandX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        ActiveCommandX();
        System.exit(0);
    }

    // ActiveCommandX function

    static void ActiveCommandX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Command cmd = null;
        Recordset rstAuthors = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";
        String strName;

        try
        {
            System.out.println("Enter an author's name (e.g., Ringer):");
            strName = in.readLine().trim();
            cmd = new Command();
```

```

cmd.setCommandText("SELECT * FROM authors WHERE au_lname =
cmd.getParameters().append(cmd.createParameter("LastName",
    AdoEnums.DataType.CHAR,
    AdoEnums.ParameterDirection.INPUT, 20, strName));
cnConn1 = new Connection();
cnConn1.open(strCnn);
cmd.setActiveConnection(cnConn1);
rstAuthors = cmd.execute(null, AdoEnums.CommandType.TEXT);
ActiveCommandXprint(rstAuthors);
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstAuthors != null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstAuthors != null)
        if (rstAuthors.getState() == 1)
            rstAuthors.close();
    // Cleanup objects before exit.
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// ActiveCommandXprint function
static void ActiveCommandXprint(Recordset rstp)
{
    // Declarations.
    BufferedReader in =
        new BufferedReader (new InputStreamReader(System.in));

```

```

String strName;

try
{
    strName = rstp.getActiveCommand().getParameters().
        getItem("LastName").getValue().toString();
    System.out.println("\nCommand text = '" +
        rstp.getActiveCommand().getCommandText() + "'");
    System.out.println("Parameter = '" + strName + "'");
    if(rstp.getBOF())
    {
        System.out.println("Name = '" + strName + "', not found.
    }
    else
    {
        System.out.println("Name = '" +
            rstp.getField("au_fname").getString() + " " +
            rstp.getField("au_lname").getString() +
            "', author ID = '" +
            rstp.getField("au_id").getString() + "'");
    }
    System.out.println("\nPress <Enter> to continue..");
    in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstp != null)
    {
        PrintProviderError(rstp.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )

```

```

{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

//.PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

}

// EndActiveCommandJ

```

See Also

[ActiveCommand Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example (VJ++)

This example uses the [ActiveConnection](#), [CommandText](#), [CommandTimeout](#), [CommandType](#), [Size](#), and [Direction](#) properties to execute a stored procedure.

```
// BeginActiveConnectionJ
import com.ms.wfc.data.*;
import java.io.*;

public class ActiveConnectionX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        ActiveConnectionX();
        System.exit(0);
    }

    //ActiveConnectionX function

    static void ActiveConnectionX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Command cmdByRoyalty = null;
        Parameter prmByRoyalty = null;
        Recordset rstByRoyalty = null;
        Recordset rstAuthors = null;

        //Declarations.
        String strCnn;
        String strAuthorID;
        String strFName;
        String strLName;
        int intRoyalty ;
        BufferedReader in = new BufferedReader
```

```

    (new InputStreamReader (System.in));
String line = null;

try
{
    // Open a connection.

    strCnn = "Provider='sqloledb';Data Source='MySqlServer';"
        + "Initial Catalog='Pubs';Integrated Security='SSPI';";
    cnConn1 = new Connection();
    cnConn1.open(strCnn, "", "", AdoEnums.CommandType.UNSPECIFIED)

    // Define a command object for stored procedure.
    cmdByRoyalty = new Command();
    cmdByRoyalty.setActiveConnection(cnConn1);
    cmdByRoyalty.setCommandText("byRoyalty");
    cmdByRoyalty.setCommandType(AdoEnums.CommandType.STOREDPROC
    cmdByRoyalty.setCommandTimeout(15);

    //Define the stored procedure's input parameter.
    System.out.println ("\nEnter Royalty : ");
    line = in.readLine().trim();
    intRoyalty = Integer.parseInt(line);
    prmsByRoyalty = new Parameter ();
    prmsByRoyalty.setType(AdoEnums.DataType.INTEGER);
    prmsByRoyalty.setSize(3);
    prmsByRoyalty.setDirection(AdoEnums.ParameterDirection.INPUT
    prmsByRoyalty.setValue(new Integer(intRoyalty));
    cmdByRoyalty.getParameters().append(prmsByRoyalty);

    // Create a recordset by executing the command.

    rstByRoyalty = cmdByRoyalty.execute();

    // Open the Authors table to get author names for display.
    rstAuthors = new Recordset ();
    rstAuthors.open("authors", strCnn,
        AdoEnums.CursorType.FORWARDONLY,
        AdoEnums.LockType.READONLY , AdoEnums.CommandType.TABLE )

    // Print current data in the recordset,
    // adding author names from Authors table.
    System.out.println("\nAuthors with " + intRoyalty +
        " percent royalty");
    while (!rstByRoyalty.getEOF())
    {
        strAuthorID = rstByRoyalty.getField("au_id").getString(
            rstAuthors.setFilter("au_id =" + strAuthorID + ""));
        strFName = rstAuthors.getField("au_fname").getString();
        strLName = rstAuthors.getField("au_lname").getString();
    }
}

```

```

        System.out.println("\t" + strAuthorID + ", " + strFName
            + " " + strLName);
        rstByRoyalty.moveNext();
    }
    System.out.println("\n\nPress <Enter> key to continue.");
    line = in.readLine();

    //Cleanup objects before exit.
    rstByRoyalty.close();
    rstAuthors.close();
    cnConn1.close();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // Check for null pointer for connection object
    if(rstByRoyalty.getActiveConnection()==null)
        System.out.println("Exception: " + ae.getMessage());
    if(rstAuthors.getActiveConnection()==null)
        System.out.println("Exception: " + ae.getMessage());
    // As passing a Recordset, check for null pointer first.
    if (rstByRoyalty != null)
    {
        PrintProviderError(rstByRoyalty.getActiveConnection());
    }
    if (rstAuthors != null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// This catch is required if input string cannot be converted
// Integer data type.
catch ( java.lang.NumberFormatException ne)
{
    System.out.println("\nException: Integer Input required.");
}
// System Read requires this catch.
catch( java.io.IOException je )
{
    PrintIOError(je);
}
finally

```

```

    {
        // Cleanup objects before exit.
        if (rstByRoyalty != null)
            if (rstByRoyalty.getState() == 1)
                rstByRoyalty.close();
        if (rstAuthors != null)
            if (rstAuthors.getState() == 1)
                rstAuthors.close();
        if (cnConn1 != null)
            if (cnConn1.getState() == 1)
                cnConn1.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i < nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

//.PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}

```

```
// EndActiveConnectionJ
```

See Also

[ActiveConnection Property](#) | [CommandText Property](#) | [CommandTimeout Property](#) | [CommandType Property](#) | [Direction Property](#) | [Size Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActualSize and DefinedSize Properties Example (VJ++)

This example uses the [ActualSize](#) and [DefinedSize](#) properties to display the defined size and actual size of a field.

```
// BeginActualSizeJ
import com.ms.wfc.data.*;
import java.io.*;

public class ActualSizeX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        ActualSizeX();
        System.exit(0);
    }

    // ActualSizeX function

    static void ActualSizeX()
    {

        // Define ADO Objects.
        Recordset rstStores = null;

        // Declarations.
        BufferedReader in = new
            BufferedReader(new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        String strStoreName;
        String strMessage;
        String strDSize, strASize;
        int intDefinedSize;
        int intActualSize;
        int intChoice = 0;

        try
        {
```

```

// Open recordset with Stores table.
rstStores = new Recordset();
rstStores.open("stores", strCnn,
    AdoEnums.CursorType.FORWARDONLY ,
    AdoEnums.LockType.READONLY ,
    AdoEnums.CommandType.TABLE);

// Loop through the Recordset displaying the contents
// of the stor_name field, the field's defined size
// and it's actual size.

while ( !(rstStores.getEOF( )) ) // continuous loop
{
    // Read data field in the variables.
    strStoreName = rstStores.getField("stor_name").getString
    intDefinedSize =
        rstStores.getField("stor_name").getDefinedSize();
    strDSize = Integer.toString(intDefinedSize);
    intActualSize = rstStores.getField
        ("stor_name").getActualSize ( );
    strASize = Integer.toString(intActualSize);

    // Display current record information.
    strMessage = "\nStore name: " + strStoreName + "\n"
        + "Defined Size : " + strDSize + "\n"
        + "Actual Size : " + strASize;

    System.out.println(strMessage);
    System.out.println("\nPress <Enter> key to continue.");
    in.readLine();
    rstStores.moveNext();
}
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // Check for null pointer for connection object.
    if (rstStores.getActiveConnection()==null)
        System.out.println("Exception: " + ae.getMessage());
    // As passing a Recordset, check for null pointer first.
    if (rstStores != null)
    {
        PrintProviderError(rstStores.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
}

```

```

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstStores != null)
        if (rstStores.getState() == 1)
            rstStores.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

//.PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"

```

```
    }  
  }  
  // EndActualSizeJ
```

See Also

[ActualSize Property](#) | [DefinedSize Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AddNew Method Example (VJ++)

This example uses the [AddNew](#) method to create a new record with the specified name.

```
// BeginAddNewJ
import com.ms.wfc.data.*;
import java.io.*;

public class AddNewX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        AddNewX();
        System.exit(0);
    }

    // AddNewX function

    static void AddNewX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstEmployees = null;

        //Declarations.
        String strCnn;
        String strID;
        String strFirstName;
        String strLastName;
        boolean booRecordAdded ;
        BufferedReader in =
            new BufferedReader (new InputStreamReader (System.in));
        String line = null;

        try
        {
            // Open a connection.
            strCnn = "Provider='sqloledb';Data Source='MySqlServer';"
                + "Initial Catalog='Pubs';Integrated Security='SSPI';";
            cnConn1 = new Connection();
            cnConn1.open(strCnn);//, "", "", AdoEnums.CommandType.UNSPECIF
```

```

// Open Employees table.
rstEmployees = new Recordset ();
rstEmployees.open("employee", cnConn1,
    AdoEnums.CursorType.KEYSET , AdoEnums.LockType.OPTIMISTI
    AdoEnums.CommandType.TABLE );

/* Get data from the user. The employee ID must be formatted
first,middle, and last initial, five numbers, then M or F t
signify the gender. For example, the employee id for Bill
Sornsins would be "B-S55555M". */
System.out.println("\nEnter employee ID :");
strID = in.readLine().trim();
System.out.println("\nEnter first name :");
strFirstName = in.readLine().trim();
System.out.println("\nEnter last name :");
strLastName = in.readLine().trim();

// Proceed only if the user actually entered something
// for both the first and last names.

if ( !(strID.compareTo("") == 0) &
    !(strFirstName.compareTo("") == 0) &
    !(strLastName.compareTo("")== 0))
{
    // Add new record.
    rstEmployees.addNew();
    rstEmployees.getField("emp_id").setString(strID);
    rstEmployees.getField("fname").setString(strFirstName);
    rstEmployees.getField("lname").setString(strLastName);

    // update the record with the new data.
    rstEmployees.update();
    booRecordAdded = true;

    // Show the newly added data.
    System.out.println("\nNew record : "
        + rstEmployees.getField("emp_id").getString()+ " "
        + rstEmployees.getField("fname").getString()+ " "
        + rstEmployees.getField("lname").getString());
}
else
{
    System.out.println("\nPlease enter an employee ID," +
        "first name, and last name.");
}

System.out.println("\n\nPress <Enter> key to continue.");
line = in.readLine();

// Delete the new record because this is a demonstration.

```

```

        cnConn1.execute("DELETE FROM employee WHERE "
            + "emp_id = '" + strID + "'");
    }
    catch( AdoException ae )
    {
        // Notify user of any errors that result from ADO.

        //Check for null pointer for connection object.
        if(rstEmployees.getActiveConnection()==null)
            System.out.println("Exception: " + ae.getMessage());
        // As passing a Recordset, check for null pointer first.
        if (rstEmployees != null)
        {
            PrintProviderError(rstEmployees.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
        System.exit(0);
    }
    // System Read requires this catch.
    catch( java.io.IOException je )
    {
        PrintIOError(je);
    }
    finally
    {
        // Cleanup objects before exit.
        if (rstEmployees != null)
            if (rstEmployees.getState() == 1)
                rstEmployees.close();
        if (cnConn1 != null)
            if (cnConn1.getState() == 1)
                cnConn1.close();
        System.exit(0);
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;

```

```

int i      = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

//.PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

}

// EndAddNewJ

```

See Also

[AddNew Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Append and CreateParameter Methods Example (VJ++)

This example uses the [Append](#) and [CreateParameter](#) methods to execute a stored procedure with an input parameter.

```
// BeginAppendJ
import com.ms.wfc.data.*;
import java.io.*;
import com.ms.com.*;

public class AppendX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        AppendX();
    }

    // AppendX function

    static void AppendX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Command cmdByRoyalty = null;
        Parameter prmByRoyalty = null;
        Recordset rstByRoyalty = null;
        Recordset rstAuthors = null;

        //Declarations.
        String strCnn;
        String strAuthorID;
        String strFName;
        String strLName;
        int intRoyalty ;
        BufferedReader in =
            new BufferedReader (new InputStreamReader (System.in));
        String line = null;
        Variant varRoyalty;

        try
        {
```

```

// Open a connection.

strCnn = "Provider='sqloledb';Data Source='MySqlServer';"
    + "Initial Catalog='Pubs';Integrated Security='SSPI';";
cnConn1 = new Connection();
cnConn1.open(strCnn);
cnConn1.setCursorLocation(AdoEnums.CursorLocation.CLIENT);

// Open command object with one parameter.
cmdByRoyalty = new Command();
cmdByRoyalty.setCommandText("byRoyalty");
cmdByRoyalty.CommandType(AdoEnums.CommandType.STOREDPROC);

//Get parameter value and append parameter.
System.out.println ("\nEnter Royalty : ");
line = in.readLine().trim();
intRoyalty = Integer.parseInt(line);
varRoyalty = new Variant(intRoyalty);
prmByRoyalty = cmdByRoyalty.createParameter ("percentage",
    AdoEnums.DataType.INTEGER,
    AdoEnums.ParameterDirection.INPUT, 4, varRoyalty);

cmdByRoyalty.getParameters().append(prmByRoyalty);
prmByRoyalty.setValue(varRoyalty);

// Create a recordset by executing the command.
cmdByRoyalty.setActiveConnection(cnConn1);
rstByRoyalty = cmdByRoyalty.execute();

// Open the Authors table to get author names for display.
rstAuthors = new Recordset ();
rstAuthors.open("authors", cnConn1,
    AdoEnums.CursorType.FORWARDONLY,
    AdoEnums.LockType.READONLY,
    AdoEnums.CommandType.TABLE );

// Print current data in the recordset,
// adding author names from Authors table.
System.out.println("\nAuthors with " + intRoyalty +
    " percent royalty");
while (!rstByRoyalty.getEOF())
{
    strAuthorID = rstByRoyalty.getField("au_id").getString(
rstAuthors.setFilter("au_id =" + strAuthorID + "");
    strFName = rstAuthors.getField("au_fname").getString();
    strLName = rstAuthors.getField("au_lname").getString();
    System.out.println("\t" + strAuthorID + ", " + strFName
        + " " + strLName);
    rstByRoyalty.moveNext();
}

```

```

        System.out.println("\n\nPress <Enter> key to continue.");
        line = in.readLine();
    }
    catch( AdoException ae )
    {
        // Notify user of any errors that result from ADO.

        // Check for null pointer for connection object.
        if(rstByRoyalty.getActiveConnection()==null)
            System.out.println("Exception: " + ae.getMessage());
        // As passing a Recordset, check for null pointer first.
        if (rstByRoyalty != null)
        {
            PrintProviderError(rstByRoyalty.getActiveConnection());
        }
        if (rstAuthors != null)
        {
            if (rstAuthors.getActiveConnection()==null)
                System.out.println("Exception: " + ae.getMessage());
            else
                PrintProviderError(rstAuthors.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }
}

// This catch is required if input string cannot be converted
// Integer data type.
catch( java.lang.NumberFormatException ne)
{
    System.out.println("\nException: Integer Number required.")
    System.exit(0);
}
// System Read requires this catch.
catch( java.io.IOException je )
{
    PrintIOError(je);
}
finally
{
    // Cleanup objects before exit.
    if (rstByRoyalty != null)
        if (rstByRoyalty.getState() == 1)
            rstByRoyalty.close();
    if (rstAuthors != null)
        if (rstAuthors.getState() == 1)

```

```

        rstAuthors.close();
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
    System.exit(0); //required here only.
}

}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

//.PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

}

// EndAppendJ

```

See Also

[Append Method](#) | [CreateParameter Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AppendChunk and GetChunk Methods Example (VJ++)

This example uses the [AppendChunk](#) and [GetChunk](#) methods to fill an image field with data from another record.

```
// BeginAppendChunkJ
import com.ms.wfc.data.*;
import java.io.*;
import com.ms.com.*;
import java.util.*;

public class AppendChunkX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        AppendChunkX();
        System.exit(0);
    }

    // AppendChunkX function

    static void AppendChunkX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstPubInfo = null;

        //Declarations.
        String strCnn;
        String strPubID;
        String strPRInfo;
        String strMessage = "";
        long lngOffset = 0;
        long lngLogoSize;
        final int conChunkSize = 100;
        byte[] varChunk = new byte[conChunkSize];
        int intCommand = 0 ;
        int intMulChunkSize,intLastChunkSize;
        Vector varLogo = new Vector();
        BufferedReader in =
```

```

    new BufferedReader(new InputStreamReader(System.in));
String line = null;
int noOfRecords;
int noOfRecordesDisplayed = 5;
int recordCount = 0;
String info;
int indexOfComma ;

try
{
    // Open a connection.
    strCnn = "Provider='sqloledb';Data Source='MySqlServer';"
        + "Initial Catalog='Pubs';Integrated Security='SSPI';";
    cnConn1 = new Connection();
    cnConn1.open(strCnn, "", "", AdoEnums.CommandType.UNSPECIFIED)

    // Open the pub_info Table.
    rstPubInfo = new Recordset();
    rstPubInfo.setCursorType(AdoEnums.CursorType.KEYSET );
    rstPubInfo.setLockType(AdoEnums.LockType.OPTIMISTIC );

    rstPubInfo.open("pub_info", cnConn1, AdoEnums.CursorType.KEY
        AdoEnums.LockType.OPTIMISTIC, AdoEnums.CommandType.TABLE
    System.out.println ("Available logos are : \n");

    noOfRecords = rstPubInfo.getRecordCount();

    // Prompt for the Logo to copy.
    for ( int i = 0; i < noOfRecords; i++)
    {
        recordCount++;
        strMessage = strMessage +
            rstPubInfo.getField("pub_id").getString() + "\n";
        indexOfComma =
            rstPubInfo.getField("pr_info").getString().indexOf(",");
        info =
            rstPubInfo.getField("pr_info").getString().substring(
                indexOfComma );
        strMessage = strMessage + info + "\n\n";
        // Display five records at a time.
        if ( recordCount == noOfRecordesDisplayed)
        {
            System.out.println(strMessage);
            System.out.println("\n\nPress <Enter> to continue..")
            line = in.readLine();
            strMessage = "";
            recordCount = 0;
        }
        rstPubInfo.moveNext();
        // Display last records and exit if last record.

```

```

        if(rstPubInfo.getEOF())
        {
            System.out.println(strMessage);
            break;
        }
    }
    System.out.println ("\nEnter the ID of a logo to copy :\n"
    strPubID = in.readLine());

    // Copy the logo to a variable in chunks.
    rstPubInfo.setFilter("pub_id = '" + strPubID + "'");
    lngLogoSize = rstPubInfo.getField("logo").getActualSize();
    while (lngOffset < lngLogoSize)
    {
        varChunk = rstPubInfo.getField("logo")
            .getByteChunk(conChunkSize);
        int i = 0;
        while (i < conChunkSize && varLogo.size() < (int)lngLogo
        {
            varLogo.addElement(new Byte(varChunk[i]));
            i++;
        }
        lngOffset = lngOffset + conChunkSize ;
    }

    //Get the data for New ID from the user.
    System.out.println
        ("\nEnter a new pub ID [must be > 9899 & < 9999]:");
    strPubID = in.readLine().trim();

    System.out.println ("\nEnter descriptive text :");
    strPRInfo = in.readLine().trim();

    //Temporarily add new publisher to publishers table to
    //avoid getting error foreign key constraint.
    cnConn1.execute("INSERT publishers(pub_id) VALUES('" +
        strPubID + "'");

    //Add a new record.
    rstPubInfo.addNew();
    rstPubInfo.getField("pub_id").setString(strPubID);
    rstPubInfo.getField("pr_info").setString(strPRInfo);

    //Copy the selected logo to the new logo in chunks.
    lngOffset = 0;

    //Divide logosize in multiples of constant chunk size.
    intMulChunkSize =
        (varLogo.size()/conChunkSize) * conChunkSize;
    intLastChunkSize = varLogo.size()- intMulChunkSize ;

```

```

byte[] arrChunk = new byte[conChunkSize];
byte[] lastChunk = new byte[intLastChunkSize];

while ( lngOffset < varLogo.size () )
{
    int ii = 0 ;
    // Copy the logo in constant chunk size.
    if ( (int)lngOffset < intMulChunkSize)
    {
        while (ii < conChunkSize &&
            (int)lngOffset < varLogo.size () )
        {
            arrChunk[ii] =
                ((Byte)varLogo.elementAt
                ((int)lngOffset)).byteValue();
            ii++;
            lngOffset++;
        }
        rstPubInfo.getField("logo").appendChunk(arrChunk);
    }
    // Copy the last remaining chunk.
    else
    {
        while (ii < intLastChunkSize &&
            (int)lngOffset < varLogo.size () )
        {
            lastChunk[ii] =
                ((Byte)varLogo.elementAt
                ((int)lngOffset)).byteValue();
            ii++;
            lngOffset++;
        }
        rstPubInfo.getField("logo").appendChunk(lastChunk)
    }
}

// Update the new recordset with new logo.

rstPubInfo.update();

//Show the newly added data.
System.out.println ("\nNew Record : " +
    rstPubInfo.getField("pub_id").getString() + "\n");
System.out.println ("Description : " +
    rstPubInfo.getField("pr_info").getString() + "\n");
System.out.println ("Logo Size : " +
    rstPubInfo.getField("logo").getActualSize() );

```

```

        System.out.println ("\n\nPress <Enter> key to continue."
        in.readLine());

        //Delete new records because this is a demonstration.
        rstPubInfo.requery();
        cnConn1.execute("DELETE FROM pub_info WHERE pub_id = '"
            strPubID + "'");
        cnConn1.execute("DELETE FROM publishers WHERE pub_id =
            strPubID + "'");
    }
    catch( AdoException ae )
    {
        // Notify user of any errors that result from ADO.

        // Check for null pointer for connection object.
        if(rstPubInfo.getActiveConnection()==null)
            System.out.println("Exception: " + ae.getMessage())
        // As passing a Recordset, check for null pointer first.
        if (rstPubInfo != null)
        {
            PrintProviderError(rstPubInfo.getActiveConnection());
            PrintADOError(ae);
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }
    // System Read requires this catch.
    catch( java.io.IOException je )
    {
        PrintIOError(je);
    }
    finally
    {
        // Cleanup objects before exit.
        if (rstPubInfo != null)
            if (rstPubInfo.getState() == 1)
                rstPubInfo.close();
        if (cnConn1 != null)
            if (cnConn1.getState() == 1)
                cnConn1.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{

```

```

// Print Provider errors from Connection object.
// ErrItem is an item object in the Connections Errors collect
com.ms.wfc.data.Error ErrItem = null;
long nCount = 0;
int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

// PrintADOError Function

static void PrintADOError(AdoException ae)
{
    System.out.println("\t Error Source = " + ae.getSource() + "\n
    System.out.println("\t Description = " + ae.getMessage() + "\n
}
}
// EndAppendChunkJ

```

See Also

[AppendChunk Method](#) | [Field Object](#) | [GetChunk Method](#)

ADO 2.5 Samples 

Attributes and Name Properties

Example (VJ++)

This example displays the value of the [Attributes](#) property for [Connection](#), [Field](#), and [Property](#) objects. It uses the [Name](#) property to display the name of each **Field** and **Property** object.

```
// BeginAttributesJ
import com.ms.wfc.data.*;
import java.io.*;

public class AttributesX
{
    // The main entry point for the application
    public static void main (String[] args)
    {
        AttributesX();
        System.exit(0);
    }

    // AttributeX Function

    static void AttributesX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstEmployees = null;
        Fields listOfFields = null;
        AdoProperties listOfProperties = null;

        //Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";
        int recordDisplaySize = 15;
        int propertyCount = 0;
        try
        {
            // Open connection and recordset.
            cnConn1 = new Connection();
            cnConn1.open(strCnn);
```

```

rstEmployees = new Recordset ();
rstEmployees.open("employee",
    cnConn1,AdoEnums.CursorType.FORWARDONLY,
    AdoEnums.LockType.READONLY, AdoEnums.CommandType.TABLE);

// Display the attributes of the connection.
System.out.println("Connection attributes = "
    + cnConn1.getAttributes());
// Display the attributes of the Employees table's fields.
System.out.println("\n\nField attributes : " + "\n");

listOfFields = rstEmployees.getFields();

for ( int i=0; i < listOfFields.getCount();i++)
{
    System.out.println("\t\t" + listOfFields.getItem(i).getN
        + " = " + listOfFields.getItem(i).getAttributes());
}

// Display fields of the Employees table which are NULLABLE
System.out.println("\n\nNULLABLE Fields : " + "\n");
for ( int i=0; i < listOfFields.getCount();i++)
{
    if ((listOfFields.getItem(i).getAttributes() &
        AdoEnums.FieldAttribute.ISNULLABLE) > 0)
        System.out.println("\t\t" +
            listOfFields.getItem(i).getName());
}

System.out.println ("\n\nPress <Enter> key to continue..");
line = in.readLine();

// Display the attributes of the Employees table's properti
System.out.println("\n\nProperty attributes : " );
listOfProperties = rstEmployees.getProperties();
for ( int i=0; i < listOfProperties.getCount() ;i++)
{
    System.out.println("\t\t" +
        listOfProperties.getItem(i).getName()
        + " = " + listOfProperties.getItem(i).getAttributes()

    if (propertyCount == recordDisplaySize)
    {
        System.out.println ("\n\nPress <Enter> key to continu
        line = in.readLine();
        propertyCount = 0;
    }
    propertyCount++;
}

```

```

    }

    System.out.println ("\n\nPress <Enter> key to continue.");
    line = in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // Check for null pointer for connection object.
    if (rstEmployees.getActiveConnection()==null)
        System.out.println("Exception: " + ae.getMessage());

    // As passing a Recordset, check for null pointer first.

    if (rstEmployees != null)
    {
        PrintProviderError(rstEmployees.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect

```

```

com.ms.wfc.data.Error ErrItem = null;
long nCount = 0;
int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i < nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

//.PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

}

// EndAttributesJ

```

See Also

[Attributes Property](#) | [Connection Object](#) | [Field Object](#) | [Name Property](#) | [Property Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

BeginTrans, CommitTrans, and RollbackTrans Methods Example (VJ++)

This example changes the book type of all psychology books in the *Titles* table of the database. After the [BeginTrans](#) method starts a transaction that isolates all the changes made to the *Titles* table, the [CommitTrans](#) method saves the changes. You can use the [Rollback](#) method to undo changes that you saved using the [Update](#) method.

```
// BeginBeginTransJ
import com.ms.wfc.data.*;
import java.io.*;

public class BeginTransX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        BeginTransX();
        System.exit(0);
    }

    // BeginTransX function
    static void BeginTransX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstTitles = null;

        //Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";
        String strTitle;
        String strType;
        BufferedReader in =
            new BufferedReader (new InputStreamReader (System.in));
        String line = null;
        String strMessage="";
        int intChoice = 0;
```

```

int recordDisplaySize = 15;
int recordCount = 0;

try
{
    // Open a connection.

    cnConn1 = new Connection();
    cnConn1.open(strCnn, "", "", AdoEnums.CommandType.UNSPECIFIED)

    // Open the Titles table.
    rstTitles = new Recordset ();
    rstTitles.setCursorType(AdoEnums.CursorType.DYNAMIC);
    rstTitles.setLockType(AdoEnums.LockType.PESSIMISTIC);
    rstTitles.open("titles", cnConn1, AdoEnums.CursorType.DYNAMIC
        AdoEnums.LockType.PESSIMISTIC , AdoEnums.CommandType.TABL

    rstTitles.moveFirst();
    cnConn1.beginTrans();

    // Loop through recordset and ask user if he/she wants
    // to change the type for a specified title.

    while (!rstTitles.getEOF())
    {
        strType = rstTitles.getField("type").getString().trim();
        if ((rstTitles.getField("type").getString().
            trim()).compareTo("psychology")== 0)
        {
            strTitle = rstTitles.getField("title").getString().tr
            System.out.println("\nTitle : " + strTitle + "\n\n"
                + "Change type to self help (1 -> Yes / 2 -> No)
            // Change the title for the specified book.
            line = in.readLine().trim();
            intChoice = Integer.parseInt(line);
            if ( intChoice == 1)
            {
                rstTitles.getField("type").setString("self_help");
                rstTitles.update();
            }
        }
        rstTitles.moveNext();
    }

    // Ask if the user wants to commit to all the
    // changes made above.
    System.out.println
        ("\n\nSave all changes (1 -> Yes / 2 -> No) ?");
    line = in.readLine().trim();
    intChoice = Integer.parseInt(line);

```

```

if ( intChoice == 1)
    cnConn1.commitTrans();
else
    cnConn1.rollbackTrans();

// Print current data in recordset.
rstTitles.requery();
rstTitles.moveFirst();

while(true)
{
    strMessage = strMessage + "\n " +
        rstTitles.getField("title").getString() + " - "
        + rstTitles.getField("type").getString() ;

    if ( recordCount == recordDisplaySize)
    {
        System.out.println(strMessage);
        System.out.println("\n\nPress <Enter> key to continue");
        line = in.readLine();
        strMessage = "";
        recordCount = 0;
    }
    recordCount++;
    rstTitles.moveNext();
    if(rstTitles.getEOF())
    {
        System.out.println(strMessage);
        break;
    }
}
System.out.println("\n\nPress <Enter> key to continue..");
line = in.readLine();

// Restore original data because this
// is a demonstration.
rstTitles.moveFirst();
while (!rstTitles.getEOF())
{
    if ((rstTitles.getField("type").getString().
        trim().compareTo("self_help"))== 0)
    {
        rstTitles.getField("type").setString("psychology");
        rstTitles.update();
    }
    rstTitles.moveNext();
}
}
catch( AdoException ae )

```

```

{
    // Notify user of any errors that result from ADO.

    // Check for null pointer for connection object.
    if(rstTitles.getActiveConnection()==null)
        System.out.println("Exception: " + ae.getMessage());

    // As passing a Recordset, check for null pointer first.
    if (rstTitles != null)
    {
        PrintProviderError(rstTitles.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System Read requires this catch.
catch( java.io.IOException je )
{
    PrintIOError(je);
}
finally
{
    // Cleanup objects before exit.
    if (rstTitles != null)
        if (rstTitles.getState() == 1)
            rstTitles.close();
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

```

// PrintProviderError Function

```

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
    }
}

```

```

        for (i = 0; i < nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }

}

//.PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

}

// EndBeginTransJ

```

See Also

[BeginTrans, CommitTrans, and RollbackTrans Methods](#) | [Update Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

BOF, EOF, and Bookmark Properties

Example (VJ++)

This example uses the [BOF](#) and [EOF](#) properties to display a message if a user tries to move past the first or last record of a [Recordset](#). It uses the [Bookmark](#) property to let the user flag a record in a **Recordset** and return to it later.

```
// BeginBOFEOFJ
import com.ms.wfc.data.*;
import java.io.*;
import com.ms.com.*;

public class BOFEOFBookmark
{
    Variant varBookmark;
    BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
    String line = null;

    // The main entry point for the application.

    public static void main (String[] args)
    {
        BOFEOFBookmark b1 = new BOFEOFBookmark ();
        b1.BOFX();
        b1.BOFX2();
        System.exit(0);
        try
        {
            b1.finalize();
        }
        catch(Throwable te)
        {
            System.out.println("Exception: " + te.getMessage());
        }
    }

    // The main entry point for the application.

    public void BOFX()
    {
        // Declarations.
        Recordset rstPublishers = null;
```

```

String strCnn;
String strMessage;
int intCommand = 0;

strCnn = "Provider='sqloledb';Data Source='MySqlServer';"
    + "Initial Catalog='Pubs';Integrated Security='SSPI'";
try
{
    // Open a recordset with data from Publishers table.
    rstPublishers = new Recordset();
    rstPublishers.setCursorType(AdoEnums.CursorType.STATIC );
    rstPublishers.setCursorLocation(
        AdoEnums.CursorLocation.CLIENT);

    // Use client cursor to enable AbsolutePosition property.

    rstPublishers.open(new String(
        "SELECT pub_id, pub_name FROM Publishers ORDER BY pub_name",
        strCnn, AdoEnums.CursorType.STATIC ,
        AdoEnums.LockType.BATCHOPTIMISTIC,
        AdoEnums.CommandType.TEXT );
    rstPublishers.moveFirst());

    //Display information about current record and get user input
    while ( true)
    {
        strMessage = "\nPublisher : " +
            rstPublishers.getField("pub_name").getString() + "\n"
            + " (Record " + rstPublishers.getAbsolutePosition()
            + " of " + rstPublishers.getRecordCount() + ")" + "\n"
            + "Enter command : " + "\n"
            + "[1 - next / 2 - previous / " + "\n"
            + "3 - set bookmark / 4 - go to bookmark] / 5 - quit]";
        System.out.println (strMessage);
        line = in.readLine();
        //No entry exits loop.
        if (line.length() == 0)
            break;
        //convert string entry to int.
        intCommand = Integer.parseInt(line);
        //out of range entry exits loop.
        if ((intCommand < 1) || (intCommand > 4)) break ;

        //Call method based on user's validated selection.
        MoveAny(intCommand, rstPublishers);
    }
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

```

```

        // As passing a Recordset, check for null pointer first.
        if (rstPublishers != null)
        {
            PrintProviderError(rstPublishers.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }

    // This catch is required if input string cannot be converted
    // Integer data type. "TCS[VSD]"

    catch ( java.lang.NumberFormatException ne)
    {
        System.out.println("\nException: Integer Input required." )
    }

    // System Read requires this catch.
    catch( java.io.IOException je )
    {
        PrintIOError(je);
    }
    finally
    {
        // Cleanup objects before exit.
        if (rstPublishers != null)
            if (rstPublishers.getState() == 1)
                rstPublishers.close();
    }
}

// MoveAny Function

public void MoveAny(int intChoice, Recordset rsTemp)
{
    // Move Forward or backward per selection from user,
    // trapping for BOF and EOF.
    try
    {
        switch(intChoice)
        {
            case 1: // Equals char of 1.
                rsTemp.moveNext();
                if (rsTemp.getEOF())
                {
                    System.out.println (

```

```

        "\nMoving past the last record \nTry again." );
        rsTemp.moveLast();
    }
    break;
case 2: // Equals char of 2.
    rsTemp.movePrevious();
    if (rsTemp.getBOF())
    {
        System.out.println (
            "\nMoving past the first record \nTry again." )
        rsTemp.moveFirst();
    }
    break;
case 3: // Equals char of 3.
    // Store the bookmark of the current record.
    varBookmark = (Variant)rsTemp.getBookmark();
    break;
case 4: // Equals char of 4.
    // Go to the record indicated by the stored bookmark.
    if (varBookmark == null)
        System.out.println ("\nNo bookmark set!");
    else
        rsTemp.setBookmark((Object)varBookmark);
    break;
default:
    break;
}
}
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rsTemp != null)
    {
        PrintProviderError(rsTemp.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect

```

```

com.ms.wfc.data.Error ErrItem = null;
long nCount = 0;
int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}
}

//.PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

////////////////////////////////////
//          BOFX2() Function.          //
////////////////////////////////////

public void BOFX2()
{
    Recordset rs = null ;

    try
    {
        // Declarations.
        rs = new Recordset();
        Variant[] arrbmk = new Variant[11];
        rs.setCursorLocation( AdoEnums.CursorLocation.CLIENT);
        String strCnn = "Provider='sqloledb';Data Source='MySQLS
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";
        rs.setActiveConnection (strCnn);
        // Open recorset with data from authors table.
        rs.open((new String("SELECT * FROM authors")),strCnn,AdoEn
        System.out.println ("\nNumber of records before filtering :

```

```

int ii = 0;

// Create array of bookmarks.
while (rs.getEOF() != true && ii < 11)
{
    arrbmk[ii] = (Variant)rs.getBookmark();
    ii++;
    rs.move (2);
}

// set Filter to recordset.
Variant bmk=new Variant();
bmk.putVariantArray(arrbmk);
rs.setFilter(bmk);
System.out.println ("\nNumber of records after filtering :

// Display the records after filtering.
rs.moveFirst();
while (!rs.getEOF())
{
    System.out.println ("\t" +rs.getAbsolutePosition() + "
    rs.moveNext();
}
rs.close();
System.out.println ("\n\nPress <Enter> key to continue.");
in.readLine();

}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rs != null)
    {
        PrintProviderError(rs.getActiveConnection());
        System.out.println("Exception: " + ae.getMessage());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System Read requires this catch.
catch( java.io.IOException je )
{
    PrintIOError(je);
}
}
}
}

```

// EndBOFE0FJ

See Also

[BOF, EOF Properties](#) | [Bookmark Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

CacheSize Property Example (VJ++)

This example uses the [CacheSize](#) property to show the difference in performance for an operation performed with and without a 30-record cache.

```
// BeginCacheSizeJ
import com.ms.wfc.data.*;
import java.io.* ;

public class CacheSizeX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        CacheSizeX();
        System.exit(0);
    }

    // CacheSizeX function

    static void CacheSizeX()
    {

        // Define ADO Objects.
        Recordset rstRoySched = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        long timeStart;
        long timeEnd;
        float timeNoCache;
        float timeCache;
        int  intLoop;
        String strTemp;

        try
        {
            // Open the RoySched table.
            rstRoySched = new Recordset();
```

```

rstRoySched.open("roysched", strCnn,
    AdoEnums.CursorType.FORWARDONLY,
    AdoEnums.LockType.READONLY,
    AdoEnums.CommandType.TABLE);

// Enumerate the Recordset object twice and record
// the elapsed time.

timeStart = System.currentTimeMillis();
for ( intLoop = 0; intLoop < 2; intLoop++)
{
    rstRoySched.moveFirst();
    while (!rstRoySched.getEOF())
    {
        // Execute a simple operation for the
        // performance test.
        strTemp = rstRoySched.getField("title_id").getString(
            rstRoySched.moveNext());
    }
}

timeEnd = System.currentTimeMillis();
timeNoCache =(float)(timeEnd - timeStart)/1000f;

// Cache records in groups of 30 records.
rstRoySched.moveFirst();
rstRoySched.setCacheSize(30);
timeStart = System.currentTimeMillis();

// Enumerate the Recordset object twice and record
// the elapsed time.
for ( intLoop = 0; intLoop < 2; intLoop++)
{
    rstRoySched.moveFirst();
    while (!rstRoySched.getEOF())
    {
        // Execute a simple operation for the
        // performance test.

        strTemp = rstRoySched.getField("title_id").getString(
            rstRoySched.moveNext());
    }
}

timeEnd = System.currentTimeMillis();
timeCache = (float)(timeEnd - timeStart)/1000f;

// Display performance results.

```

```

        System.out.println("\nCaching Performance Results:");
        System.out.println("\n\tNo Cache: " + timeNoCache + " secon
        System.out.println("\n\t30-record cache: " + timeCache +
            " seconds");
        System.out.println("\n\nPress <Enter> to continue..");
        in.readLine();
    }
    catch( AdoException ae )
    {
        // Notify user of any errors that result from ADO.

        // Check for null pointer for connection object.
        if (rstRoySched.getActiveConnection()==null)
            System.out.println("Exception: " + ae.getMessage());

        // As passing a Recordset, check for null pointer first.
        if (rstRoySched != null)
        {
            PrintProviderError(rstRoySched.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }

    // System read requires this catch.
    catch( java.io.IOException je)
    {
        PrintIOError(je);
    }
    finally
    {
        // Cleanup objects before exit.
        if (rstRoySched != null)
            if (rstRoySched.getState() == 1)
                rstRoySched.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;

```

```

long nCount = 0;
int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

}

// EndCacheSizeJ

```

See Also

[CacheSize Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Cancel Method Example (VJ++)

This example uses the [Cancel](#) method to cancel a command executing on a [Connection](#) object if the connection is busy.

```
// BeginCancelJ
import com.ms.wfc.data.*;
import java.io.* ;

public class CancelX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        CancelX();
        System.exit(0);
    }

    // CancelX function

    static void CancelX()
    {
        // Define command strings.
        String strCmdChange = "UPDATE titles SET type = 'self_help' "
            + "WHERE type = 'psychology'";
        String strCmdRestore = "UPDATE titles SET type = 'psychology'
            + "WHERE type = 'self_help'";
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        // Define ADO Objects.
        Connection cnConn1 = null;

        //Declarations.
        boolean booChanged = false;
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;

        try
        {
            // Open a connection.
            cnConn1 = new Connection();
            cnConn1.open(strCnn);
```

```

// Begin a transaction, then execute a command asynchronous
cnConn1.beginTrans();
cnConn1.execute(strCmdChange,
    AdoEnums.ExecuteOption.ASYNCEXECUTE);

// do something else for a little while - this could be cha
for (int intLoop = 0; intLoop < 10; intLoop++)
{
    System.out.println(intLoop);
}

// If the command has NOT completed, cancel the execute
// and roll back the transaction. Otherwise, commit the
// transaction.

if ((cnConn1.getState() & AdoEnums.ObjectState.EXECUTING) >
{
    cnConn1.cancel();
    cnConn1.rollbackTrans();
    booChanged = false;
    System.out.println("\nUpdate canceled.");
}
else
{
    cnConn1.commitTrans();
    booChanged = true;
    System.out.println("\nUpdate complete.");
}

//If the change was made, restore the data
// because this is a demonstration.

if(booChanged )
{
    cnConn1.execute(strCmdRestore);
    System.out.println("\nData restored.");
}

System.out.println("\n\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a connection, check for null pointer first.
    if (cnConn1 != null)
    {

```

```

        PrintProviderError(cnConn1);
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}
finally
{
    // Cleanup objects before exit.
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

```

```
static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}

// EndCancelJ
```

See Also

[Cancel Method](#) | [Connection Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Clone Method Example (VJ++)

This example uses the [Clone](#) method to create copies of a [Recordset](#) and then lets the user position the record pointer of each copy independently.

```
// BeginCloneJ
import com.ms.wfc.data.*;
import java.io.* ;

public class CloneX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        CloneX();
        System.exit(0);
    }

    // CloneX function

    static void CloneX()
    {
        // Assign SQL statement and connection string to variables.
        String strSQL = "SELECT stor_name FROM Stores "
            + "ORDER BY stor_name";

        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        // Define ADO Objects.
        Recordset[] arstStores = null;

        //Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strMessage;
        String strFind;
        int intLoop;
        boolean booExit = true;

        try
        {
            // Open recordset as a static cursor type recordset.
```

```

arstStores = new Recordset[3];
arstStores[0] = new Recordset();
arstStores[0].setCursorType(AdoEnums.CursorType.STATIC);
arstStores[0].setLockType(AdoEnums.LockType.BATCHOPTIMISTIC);
arstStores[0].open(strSQL, strCnn, AdoEnums.CursorType.STATIC
    AdoEnums.LockType.BATCHOPTIMISTIC, AdoEnums.CommandType.T

// Create two clones of the original Recordset.
arstStores[1] = (Recordset)arstStores[0].clone
    (AdoEnums.LockType.BATCHOPTIMISTIC);
arstStores[2] = (Recordset)arstStores[0].clone
    (AdoEnums.LockType.BATCHOPTIMISTIC);

while(booExit)
{
    // Loop through the array so that on each pass,
    // the user is searching a different copy of the
    // same Recordset.
    for (intLoop = 0; intLoop < 3; intLoop++)
    {
        // Ask for search string while showing where
        // the current record pointer is for each Recordset
        strMessage = "\nRecordsets from stores table:" + "\n"
            + " 1 - Original - Record pointer at "
            + arstStores[0].getField("stor_name").getString()
            + "\n" + " 2 - Clone - Record pointer at "
            + arstStores[1].getField("stor_name").getString()
            + "\n" + " 3 - Clone - Record pointer at "
            + arstStores[2].getField("stor_name").getString()
            + "\n";
        System.out.println(strMessage);
        System.out.println("Enter search string for #"
            + (intLoop+1) + "(Press <Enter> to Exit.)");

        strFind = in.readLine().trim();
        if(strFind.length() == 0)
        {
            booExit = false;
            break;
        }

        // Find the search string; if there's no
        // match, jump to the last record.
        arstStores[intLoop].setFilter("stor_name >= '" +
            strFind + "'");
        if (arstStores[intLoop].getEOF())
        {
            arstStores[intLoop].setFilter
                (new Integer(AdoEnums.FilterGroup.NONE));
            arstStores[intLoop].moveLast();
        }
    }
}

```

```

        }
    }
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a connection, check for null pointer first.
    if (arstStores[0] != null)
    {
        PrintProviderError(arstStores[0].getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (arstStores[0] != null)
        if (arstStores[0].getState() == 1)
            arstStores[0].close();
    if (arstStores[1] != null)
        if (arstStores[1].getState() == 1)
            arstStores[1].close();
    if (arstStores[2] != null)
        if (arstStores[2].getState() == 1)
            arstStores[2].close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;
}

```

```

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndCloneJ

```

See Also

[Clone Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

CompareBookmarks Method Example (VJ++)

This example demonstrates the [CompareBookmarks](#) method. The relative value of bookmarks is seldom needed unless a particular bookmark is somehow special.

Designate a random row of a [Recordset](#) derived from the *Authors* table as the target of a search. Then display the position of each row relative to that target.

```
// BeginCompareBookmarksJ
import com.ms.wfc.data.*;
import java.io.* ;
import com.ms.com.*;

public class CompareBookmarksX //
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        CompareBookmarksX();
        System.exit(0);
    }

    // CompareBookmarksX function

    static void CompareBookmarksX()
    {
        // Define ADO Objects.
        Recordset rstAuthors = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'";
        int intCount;
        Variant varTarget = null;
        int intResult;
        String strAns;
        int intDisplaySize = 15;
```

```

try
{
    rstAuthors = new Recordset();
    rstAuthors.open("SELECT * FROM authors",
        strCnn,
        AdoEnums.CursorType.STATIC,
        AdoEnums.LockType.READONLY,
        AdoEnums.CommandType.TEXT);
    intCount = rstAuthors.getRecordCount();
    System.out.println("Rows in the Recordset = " +
        Integer.toString(intCount));

    // Exit if an empty recordset.
    if(intCount == 0)
        System.exit(0);

    // Randomize.
    intCount = (int)(intCount * Math.random());
    // Get position between 0 and count-1.
    System.out.println("\nRandomly chosen row position = " +
        Integer.toString(intCount)+ "\n");
    rstAuthors.move(intCount, new Integer(AdoEnums.Bookmark.FIRST));
    varTarget = (Variant)rstAuthors.getBookmark();
    // Remember the mystery row.
    intCount = 0;
    rstAuthors.moveFirst();

    // Loop through recordset.
    while(!rstAuthors.getEOF())
    {
        intResult = rstAuthors.compareBookmarks
            ((Variant)rstAuthors.getBookmark(), varTarget);
        if(intResult == AdoEnums.Compare.NOTEQUAL)
            System.out.println("Row " +
                Integer.toString(intCount) +
                ": Bookmarks are not equal.");
        else if(intResult == AdoEnums.Compare.NOTCOMPARABLE)
            System.out.println("Row " +
                Integer.toString(intCount) +
                ": Bookmarks are not comparable.");
        else
        {
            switch(intResult)
            {
                case AdoEnums.Compare.LESSTHAN :
                    strAns = "less than";
                    break;
                case AdoEnums.Compare.EQUAL :
                    strAns = "equal to";
                    break;
            }
        }
    }
}

```

```

        case AdoEnums.Compare.GREATERTHAN :
            strAns = "greater than";
            break;
        default :
            strAns = "in error comparing to";
            break;
    }
    System.out.println("Row position " +
        Integer.toString(intCount) +
        " is " + strAns + " the target.");
}
if(intCount % intDisplaySize == 0 && intCount > 0)
{
    System.out.println("\nPress <Enter> to continue..");
    in.readLine();
}
intCount++;
rstAuthors.moveNext();
}

System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstAuthors != null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstAuthors != null)
        if (rstAuthors.getState() == 1)

```

```

        rstAuthors.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

}

// EndCompareBookmarksJ

```

See Also

[CompareBookmarks Method](#) | [Recordset Object](#)

ADO 2.5 Samples 

ConnectionString, ConnectionTimeout, and State Properties Example (VJ++)

This example demonstrates different ways of using the [ConnectionString](#) property to open a [Connection](#) object. It also uses the [ConnectionTimeout](#) property to set a connection timeout period, and the [State](#) property to check the state of the connections. The GetState function is required for this procedure to run.

```
// BeginConnectionStringJ
import com.ms.wfc.data.*;
import java.io.* ;

public class ConnectionStringX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        ConnectionStringX();
        System.exit(0);
    }

    // ConnectionStringX function

    static void ConnectionStringX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Connection cnConn2 = null;
        Connection cnConn3 = null;
        Connection cnConn4 = null;

        //Declarations.

        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strTemp;
```

```

try
{
    // Open a connection using OLE DB syntax.
    cnConn1 = new Connection();
    cnConn1.setConnectionString(
        "Provider='sqloledb';Data Source='MySqlServer';" +
        "Initial Catalog='Pubs';Integrated Security='SSPI'");
    cnConn1.setCommandTimeout(30);
    cnConn1.open();
    strTemp = getState(cnConn1.getState());
    System.out.println("CnConn1 state: " + strTemp);

    // Open a connection using a DSN and ODBC tags.
    // It is assumed that you have create DSN 'Pubs' with a use
    // 'MyUserId' and password as 'MyPassword'.
    cnConn2 = new Connection();
    cnConn2.setConnectionString("DSN='Pubs';UID='MyUserId';PWD="
    cnConn2.open();
    strTemp = getState(cnConn2.getState());
    System.out.println("CnConn2 state: " + strTemp);

    // Open a connection using a DSN and OLE DB tags.
    cnConn3 = new Connection();
    cnConn3.setConnectionString
        ("Data Source='Pubs'");
    cnConn3.open();
    strTemp = getState(cnConn3.getState());
    System.out.println("CnConn3 state: " + strTemp);

    // Open a connection using a DSN and individual
    // arguments instead of a connection string.
    // It is assumed that you have create DSN 'Pubs' with a use
    // 'MyUserId' and password as 'MyPassword'.
    cnConn4 = new Connection();
    cnConn4.open("Pubs", "MyUserId", "MyPassword");
    strTemp = getState(cnConn4.getState());
    System.out.println("CnConn4 state: " + strTemp);

    System.out.println("\n\nPress <Enter> to continue..");
    in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    System.out.println("Exception: " + ae.getMessage());
}

```

```

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
    if (cnConn2 != null)
        if (cnConn2.getState() == 1)
            cnConn2.close();
    if (cnConn3 != null)
        if (cnConn3.getState() == 1)
            cnConn3.close();
    if (cnConn4 != null)
        if (cnConn4.getState() == 1)
            cnConn4.close();
}

}

// getState Function

static String getState(int intState)
{
    // Returns current state of the connection object.
    String strState=null;

    switch(intState)
    {
    case AdoEnums.ObjectState.CLOSED :
        strState = new String("adStateClosed");
        break;
    case AdoEnums.ObjectState.OPEN :
        strState = new String("adStateOpen");
        break;
    default :
        break;
    }
    return strState;
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)

```

```
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndConnectionStringJ
```

See Also

[Connection Object](#) | [ConnectionString Property](#) | [ConnectionTimeout Property](#) | [State Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Count Property Example (VJ++)

This example demonstrates the [Count](#) property with two collections in the *Employees* database. The property obtains the number of objects in each collection, and sets the upper limit for loops that enumerate these collections. Another way to enumerate these collections without using the **Count** property would be to use For Each...Next statements.

```
// BeginCountJ
import com.ms.wfc.data.*;
import java.io.* ;

public class CountX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        CountX();
        System.exit(0);
    }

    // CountX function

    static void CountX()
    {

        // Define ADO Objects.
        Recordset rstEmployees = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer '
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        int intLoop;
        int intDisplaySize = 20;
        int recCount=0;

        try
        {
            rstEmployees = new Recordset();
```

```

// Open recordset with data from Employees table.
rstEmployees.open("employee", strCnn,
    AdoEnums.CursorType.FORWARDONLY,
    AdoEnums.LockType.READONLY,
    AdoEnums.CommandType.TABLE);

// Print information about Fields collection.
System.out.println(rstEmployees.getFields().getCount() +
    " Fields in Employees");
for ( intLoop = 0; intLoop <
    rstEmployees.getFields().getCount(); intLoop++)
{
    System.out.println("\t" +
        rstEmployees.getFields().getItem(intLoop).getName());
}
System.out.println("\n\nPress <Enter> to continue..");
in.readLine();

// Print information about Properties collection.
System.out.println(rstEmployees.getProperties().getCount()
    " Properties in Employees");
for ( intLoop = 0; intLoop <
    rstEmployees.getProperties().getCount(); intLoop++)
{
    System.out.println("\t" +
        rstEmployees.getProperties().getItem(intLoop).getName
        recCount++;
    if ( recCount >= intDisplaySize)
    {
        System.out.println("\n\nPress <Enter> to continue..")
        in.readLine();
        recCount = 0;
    }
}
System.out.println("\n\nPress <Enter> to continue..");
in.readLine();

}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // Check for null pointer for connection object.
    if (rstEmployees.getActiveConnection()==null)
    {
        System.out.println("Exception: " + ae.getMessage());
    }
    else
    {

```

```

        // As passing a Recordset, check for null pointer first.
        if (rstEmployees != null)
        {
            PrintProviderError(rstEmployees.getActiveConnection())
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

```

```
    }  
}  
  
// PrintIOError Function  
  
static void PrintIOError( java.io.IOException je)  
{  
    System.out.println("Error \n");  
    System.out.println("\tSource = " + je.getClass() + "\n");  
    System.out.println("\tDescription = " + je.getMessage() + "\n")  
}  
}  
  
// EndCountJ
```

See Also

[Count Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

CursorType, LockType, and EditMode Properties Example (VJ++)

This example demonstrates setting the [CursorType](#) and [LockType](#) properties before opening a [Recordset](#). It also shows the value of the [EditMode](#) property under various conditions. The EditModeOutput function is required for this procedure to run.

```
// BeginEditModeJ
import com.ms.wfc.data.*;
import java.io.* ;

public class EditModeX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        EditModeX ();
        System.exit(0);
    }

    // EditModeX function

    static void EditModeX ()
    {

        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstEmployees = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        try
        {
```

```

// Open recordset with data from Employees table.

cnConn1 = new Connection();
cnConn1.open(strCnn);
rstEmployees = new Recordset();
rstEmployees.setActiveConnection(cnConn1);
rstEmployees.setCursorLocation(AdoEnums.CursorLocation.CLIE
rstEmployees.setCursorType(AdoEnums.CursorType.STATIC);
rstEmployees.setLockType(AdoEnums.LockType.BATCHOPTIMISTIC)
rstEmployees.open("employee", cnConn1,
    AdoEnums.CursorType.STATIC,
    AdoEnums.LockType.BATCHOPTIMISTIC,
    AdoEnums.CommandType.TABLE);

// Show the EditMode property under different editing state

rstEmployees.addNew();
rstEmployees.getField("emp_id").setString("T-T5555M");
rstEmployees.getField("fname").setString("temp_fname");
rstEmployees.getField("lname").setString("temp_lname");
EditModeOutput("After AddNew:", rstEmployees.getEditMode())
rstEmployees.updateBatch();
EditModeOutput("After Update:", rstEmployees.getEditMode())
rstEmployees.getField("fname").setString("test");
EditModeOutput("After Edit:", rstEmployees.getEditMode());

System.out.println("\n\nPress <Enter> to continue..");
in.readLine();

// Delete new record because this is a demonstration.
cnConn1.execute(
    "DELETE FROM employee WHERE emp_id = 'T-T5555M'");

// Cleanup objects before exit.
rstEmployees.close();
cnConn1.close();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // Check for null pointer for connection object.
    if (cnConn1==null)
        System.out.println("Exception: " + ae.getMessage());

    // As passing a Recordset, check for null pointer first.
    if (rstEmployees != null)
    {
        PrintProviderError(rstEmployees.getActiveConnection());
    }
}

```

```

    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
}
}

// EditModeOutput Function

static void EditModeOutput(String strTemp, int intEditMode)
{
    String strMessage="";
    // Print report based on the value of the EditMode
    // property.

    System.out.println("\n" + strTemp);
    strMessage = "\n\tEditMode = ";
    switch(intEditMode)
    {
    case AdoEnums.EditMode.NONE :
        strMessage+="adEditNone";
        break;
    case AdoEnums.EditMode.INPROGRESS :
        strMessage+="adEditInProgress";
        break;
    case AdoEnums.EditMode.ADD :
        strMessage+="adEditAdd";
        break;
    default :
        break;
    }
    System.out.println(strMessage);
}

```

```

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndEditModeJ

```

See Also

[CursorType Property](#) | [EditMode Property](#) | [LockType Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Delete Method Example (VJ++)

This example uses the [Delete](#) method to remove a specified record from a [Recordset](#).

```
// BeginDeleteJ
// The WFC class includes the ADO objects.

import com.ms.wfc.data.*;
import java.io.* ;

public class DeleteX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        DeleteX();
        System.exit(0);
    }

    // DeleteX function

    static void DeleteX()
    {

        // Define ADO Objects.
        Recordset rstRoySched = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer '
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        String strMessage="";
        String strTitleID;
        int intLoRange =0;
        int intHiRange =0;
        int intRoyalty =0;
        boolean bolFound;

        try
        {
```

```

rstRoySched = new Recordset();
rstRoySched.setCursorLocation(AdoEnums.CursorLocation.CLIENT);
rstRoySched.setCursorType(AdoEnums.CursorType.STATIC);
rstRoySched.setLockType(AdoEnums.LockType.BATCHOPTIMISTIC);

// Open RoySched table.
rstRoySched.open("SELECT * FROM roysched " +
    "WHERE royalty = 20", strCnn,
    AdoEnums.CursorType.STATIC,
    AdoEnums.LockType.BATCHOPTIMISTIC,
    AdoEnums.CommandType.TEXT);

// Prompt for a record to delete.
strMessage = "Before delete there are "
    + rstRoySched.getRecordCount()
    + " titles with 20 percent royalty:" + "\n\n";
while(!rstRoySched.getEOF())
{
    strMessage += rstRoySched.getField("title_id").getString()
        + "\n\n";
    rstRoySched.moveNext();
}
strMessage += "Enter the ID of a record to delete:" + "\n";
System.out.println(strMessage);
strTitleID = in.readLine().trim().toUpperCase();

// Move to the record and save data so it can be restored.
rstRoySched.setFilter("title_id = '" + strTitleID + "'");
if(!(rstRoySched.getRecordCount()==0))
{
    intLoRange = rstRoySched.getField("lorange").getInt();
    intHiRange = rstRoySched.getField("hirange").getInt();
    intRoyalty = rstRoySched.getField("royalty").getInt();
    bolFound = true;
}
else
{
    System.out.println("\nIncorrect ID. No Record deleted.");
    bolFound = false;
}

// Delete the record.
if(bolFound)
{
    rstRoySched.delete();
    rstRoySched.updateBatch();
}
// Show the results.
rstRoySched.setFilter(new Integer(AdoEnums.FilterGroup.NONE));
rstRoySched.requery();

```

```

strMessage = "";
strMessage = "\n\nAfter delete there are "
    + rstRoySched.getRecordCount()
    + " titles with 20 percent royalty:" + "\n\n";
while(!rstRoySched.getEOF())
{
    strMessage += rstRoySched.getField("title_id").getString
        "\n\n";
    rstRoySched.moveNext();
}
System.out.println(strMessage);
System.out.println("\nPress <Enter> to continue..");
in.readLine();

// Restore the data because this is a demonstration.
if(bolFound)
{
    rstRoySched.addNew();
    rstRoySched.getField("title_id").setString(strTitleID);
    rstRoySched.getField("lorange").setInt(intLoRange);
    rstRoySched.getField("hirange").setInt(intHiRange);
    rstRoySched.getField("royalty").setInt(intRoyalty);
    rstRoySched.updateBatch();
}
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstRoySched != null)
    {
        PrintProviderError(rstRoySched.getActiveConnection())
        System.out.println("Exception: " + ae.getMessage());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally

```

```

    {
        // Cleanup objects before exit.
        if (rstRoySched != null)
            if (rstRoySched.getState() == 1)
                rstRoySched.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i < nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

}

// EndDeleteJ

```

See Also

[Delete Method \(ADO Recordset\)](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState Properties Example (VJ++)

This example triggers an error, traps it, and displays the [Description](#), [HelpContext](#), [HelpFile](#), [NativeError](#), [Number](#), [Source](#), and [SQLState](#) properties of the resulting [Error](#) object.

```
// BeginDescriptionJ
// The WFC class includes the ADO objects.

import com.ms.wfc.data.*;
import java.io.* ;

public class DescriptionX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        DescriptionX();
        System.exit(0);
    }

    // DescriptionX function

    static void DescriptionX()
    {
        // Declarations.
        BufferedReader in = new
            BufferedReader(new InputStreamReader(System.in));

        // Define ADO Objects.
        Connection cnConn1 = null;

        try
        {
            // Intentionally trigger an error.
            cnConn1 = new Connection();
            cnConn1.open("nothing");
        }
        catch( AdoException ae )
```

```

    {
        // Notify user of any errors that result from ADO.
        PrintProviderError(cnConn1);
    }

    try
    {
        System.out.println("\nPress <Enter> key to continue.");
        in.readLine();
    }
    // System read requires this catch.
    catch( java.io.IOException je)
    {
        PrintIOError(je);
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
}

```

```
        System.out.println("\tDescription = " + je.getMessage() + "\n"  
    }  
}  
// EndDescriptionJ
```

See Also

[Description Property](#) | [Error Object](#) | [HelpContext Property](#) | [HelpFile Property](#) | [NativeError Property](#) | [Number Property](#) | [Source Property \(ADO Error\)](#) | [SQLState Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Execute, Requery, and Clear Methods Example (VJ++)

This example demonstrates the **Execute** method when run from both a [Command](#) object and a [Connection](#) object. It also uses the [Requery](#) method to retrieve current data in a recordset, and the [Clear](#) method to clear the contents of the [Errors](#) collection. The ExecuteCommand and PrintOutput procedures are required for this procedure to run.

```
// BeginExecuteJ
// The WFC class includes the ADO objects.
import com.ms.wfc.data.*;
import java.io.*;

public class ExecuteX
{
    // Main Function

    public static void main (String[] args)
    {
        ExecuteX();
    }

    // ExecuteX Function

    static void ExecuteX()
    {
        // Define string variables.
        String strSQLChange = "UPDATE Titles SET Type = "
            + "'self_help' WHERE Type = 'psychology'";
        String strSQLRestore = "UPDATE Titles SET Type = "
            + "'psychology' WHERE Type = 'self_help'";
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        // Define ADO objects.
        Connection cnConn1 = null;
        Command cmdChange = null;
        Recordset rsTitles = null;

        try
        {
            // Open connection.
```

```

cnConn1 = new Connection ();
cnConn1.open(strCnn, "", "", AdoEnums.CommandType.UNSPECIFI

// Create command object.
cmdChange = new Command();
cmdChange.setActiveConnection (cnConn1);
cmdChange.setCommandText (strSQLChange);

// Open recordset with Titles table.
rsTitles = new Recordset();
rsTitles.open("Titles", cnConn1,
              AdoEnums.CursorType.STATIC,
              AdoEnums.LockType.OPTIMISTIC,
              AdoEnums.CommandType.TABLE);

// Print report of original data.
System.out.println("\n\n\tData in Titles table "
                  + "before executing the query: \n");
PrintOutput(rsTitles);

// Clear extraneous errors from the Errors collection.
cnConn1.getErrors().clear();

// Call the ExecuteCommand subroutine to
// execute cmdChange command.
ExecuteCommand(cmdChange, rsTitles);

// Print report of new data.
System.out.println("\n\n\tData in Titles table after "
                  + "executing the query: \n");
PrintOutput(rsTitles);

// Use the Connection object's execute method to
// execute SQL statement to restore data.
cnConn1.execute(strSQLRestore);

// Print report of restored data.
System.out.println("\n\n\tData after executing the query "
                  + "to restore the original information: \n");
PrintOutput(rsTitles);
} // End Try statement.
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rsTitles != null)
    {
        PrintProviderError(rsTitles.getActiveConnection());
    }
}

```

```

        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }

    finally
    {
        // Cleanup objects before exit.
        if (rsTitles != null)
            if (rsTitles.getState() == 1)
                rsTitles.close();
        if (cnConn1 != null)
            if (cnConn1.getState() == 1)
                cnConn1.close();
    }
}

// ExecuteCommand Function

static void ExecuteCommand(Command cmdTemp, Recordset rstTemp)
{
    try
    {
        // CommandText property already set before function was cal
        cmdTemp.setCommandType(AdoEnums.CommandType.TEXT);
        cmdTemp.execute();

        // Retrieve the current data by requerying the recordset.
        rstTemp.requery(AdoEnums.CommandType.UNKNOWN);
    }
    catch( AdoException ae )
    {
        // Notify user of any errors that result from ADO.
        PrintProviderError(rstTemp.getActiveConnection());
    }
}

// PrintOutput Function

static void PrintOutput(Recordset rstTemp)
{
    // Declarations.
    BufferedReader in = new
        BufferedReader(new InputStreamReader(System.in));

    // Ensure at top of recordset.
    rstTemp.moveFirst();
}

```

```

// If EOF is true, then no data and skip print loop.
if( rstTemp.getEOF() )
{
    System.out.println("\tRecordset empty\n");
}
else
{
    // Enumerate Recordset and print data from each.
    while( !(rstTemp.getEOF()) )
    {
        // Convert variant string to convertible string type.
        System.out.println("\t"
            + rstTemp.getFields().getItem("Title").getValue() + "
            + rstTemp.getFields().getItem("Type").getValue() + "\
            rstTemp.moveNext();
    }
}
try
{
    System.out.println("\nPress <Enter> key to continue.");
    in.readLine();
}
// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

```

```
        }  
    }  
  
    // PrintIOError Function  
  
    static void PrintIOError( java.io.IOException je)  
    {  
        System.out.println("Error \n");  
        System.out.println("\tSource = " + je.getClass() + "\n");  
        System.out.println("\tDescription = " + je.getMessage() + "\n")  
    }  
}  
// EndExecuteJ
```

See Also

[Clear Method](#) | [Command Object](#) | [Connection Object](#) | [Errors Collection](#) | [Execute Method \(ADO Command\)](#) | [Execute Method \(ADO Connection\)](#) | [Requery Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Filter and RecordCount Properties Example (VJ++)

This example uses the [Filter](#) property to open a new [Recordset](#) based on a specified condition applied to an existing **Recordset**. It uses the [RecordCount](#) property to show the number of records in the two **Recordsets**. The FilterField function is required for this procedure to run.

```
// BeginFilterJ
// The WFC class includes the ADO objects.

import com.ms.wfc.data.*;
import java.io.* ;

public class FilterX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        FilterX();
        FilterX2();
        System.exit(0);
    }

    // FilterX function
    static void FilterX()
    {

        // Define ADO Objects.
        Recordset rstPublishers = null;
        Recordset rstPublishersCountry = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        int intPublisherCount;
        String strCountry;
        String strMessage;
```

```

try
{
    rstPublishers = new Recordset();

    // Open recordset with data from Publishers table.
    rstPublishers.setCursorType(AdoEnums.CursorType.STATIC);
    rstPublishers.open("publishers", strCnn,
        AdoEnums.CursorType.STATIC,
        AdoEnums.LockType.READONLY,
        AdoEnums.CommandType.TABLE);

    // Populate the Recordset.
    intPublisherCount = rstPublishers.getRecordCount();

    // Get user input.
    System.out.println("Enter a country to filter on:");
    strCountry = in.readLine().trim();

    if(!strCountry.equals(""))
    {
        // Open a filtered Recordset object.
        rstPublishersCountry =
            FilterField(rstPublishers, "Country", strCountry);
        if(rstPublishersCountry.getRecordCount()==0)
            System.out.println("\nNo publishers from that country
else
        {
            // Print number of records for the original
            // Recordset object and the filtered Recordset
            // object.
            strMessage = "\nOrders in original recordset: " + "\n
                + intPublisherCount + "\n"
                + "Orders in filtered recordset (Country = '"
                + strCountry + "'): \n"
                + rstPublishersCountry.getRecordCount();
            System.out.println(strMessage);
        }
        rstPublishersCountry.close();
    }
    System.out.println("\n\nPress <Enter> to continue..");
    in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstPublishers != null)

```

```

        {
            PrintProviderError(rstPublishers.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }

    // System read requires this catch.
    catch( java.io.IOException je)
    {
        PrintIOError(je);
    }

    finally
    {
        // Cleanup objects before exit.
        if (rstPublishers != null)
            if (rstPublishers.getState() == 1)
                rstPublishers.close();
    }
}

// FilterField Function

static Recordset FilterField(Recordset rstTemp,String strField,
                             String strFilter)
{
    // Set a filter on the specified Recordset object and then
    // open a new Recordset object.
    rstTemp.setFilter(strField + " = '" + strFilter + "'");
    return rstTemp;
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();
}

```

```

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}

// FilterX2 function

static void FilterX2()
{
    // Define ADO Objects.
    Recordset rstPublishers = null;

    // Declarations.
    BufferedReader in =
        new BufferedReader (new InputStreamReader(System.in));
    String line = null;
    String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
        + "Initial Catalog='Pubs';Integrated Security='SSPI'";

    try
    {
        rstPublishers = new Recordset();

        // Open recordset with data from Publishers table.
        rstPublishers.setCursorType(AdoEnums.CursorType.STATIC);
        rstPublishers.open("SELECT * FROM publishers " +
            "WHERE Country = 'USA'", strCnn,
            AdoEnums.CursorType.STATIC,
            AdoEnums.LockType.READONLY,
            AdoEnums.CommandType.TEXT);
    }
}

```

```

// Print current data in recordset.
rstPublishers.moveToFirst();
while(!rstPublishers.getEOF())
{
    System.out.println(rstPublishers.getField("pub_name").get
        +", "
        + rstPublishers.getField("country").getStrin
    rstPublishers.moveToNext();
}
System.out.println("\n\nPress <Enter> to continue..");
in.readLine();

// Cleanup objects before exit.
rstPublishers.close();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // Check for null pointer for connection object.
    if (rstPublishers.getActiveConnection()==null)
    {
        System.out.println("Exception: " + ae.getMessage());
    }
    else
    {
        // As passing a Recordset, check for null pointer first.
        if (rstPublishers != null)
        {
            PrintProviderError(rstPublishers.getActiveConnection(
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

}
}
// EndFilterJ

```

See Also

[Filter Property](#) | [RecordCount Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Find Method Example (VJ++)

This example uses the [Recordset](#) object's [Find](#) method to locate and count the number of business titles in the *Pubs* database. The example assumes the underlying [provider](#) does not support similar functionality.

```
// BeginFindJ
// The WFC class includes the ADO objects.
import com.ms.wfc.data.*;
import java.io.* ;
import com.ms.com.*;

public class FindX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        FindX();
        System.exit(0);
    }

    // FindX function

    static void FindX()
    {

        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstTitles = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'";
        Variant varMark = null;
        int intCount = 0;

        try
        {
            cnConn1 = new Connection();
            cnConn1.open(strCnn);
            rstTitles = new Recordset();
            rstTitles.open("SELECT title_id FROM titles",
```

```

        cnConn1,
        AdoEnums.CursorType.STATIC,
        AdoEnums.LockType.READONLY,
        AdoEnums.CommandType.TEXT);

// The default parameters are sufficient to search forward
// through a Recordset.
rstTitles.find("title_id LIKE 'BU%'");

// Skip current record to avoid finding the same row repeat
// The bookmark is redundant because Find searches from cur
// position.
while(!rstTitles.getEOF()) // Continue if last find succeed
{
    System.out.println("Title ID: "
        + rstTitles.getField("title_id").getStrin
intCount++; // Count the last title found.
varMark = (Variant)rstTitles.getBookmark();
// Note current position.
rstTitles.find("title_id LIKE 'BU'",
    1,
    AdoEnums.SearchDirection.FORWARD,
    varMark);
}

System.out.println("\nThe number of business titles is " +
    Integer.toString(intCount));
System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstTitles != null)
    {
        PrintProviderError(rstTitles.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

```

```

finally
{
    // Cleanup objects before exit.
    if (rstTitles != null)
        if (rstTitles.getState() == 1)
            rstTitles.close();
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}

```

```
// EndFindJ
```

See Also

[Find Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

GetRows Method Example (VJ++)

This example uses the [GetRows](#) method to retrieve a specified number of rows from a [Recordset](#) and to fill an array with the resulting data. The **GetRows** method will return less than the desired number of rows in two cases: either if [EOF](#) has been reached, or if **GetRows** tried to retrieve a record that was deleted by another user. The function returns **False** only if the second case occurs. The `GetRowsOK` function is required for this procedure to run.

```
// BeginGetRowsJ
// The WFC class includes the ADO objects.

import com.ms.wfc.data.*;
import java.io.* ;
import com.ms.com.*;

public class GetRowsX
{
    // The main entry point for the application.

    static Variant avarRecords = null;

    public static void main (String[] args)
    {
        GetRowsX();
        System.exit(0);
    }

    // GetRowsX function

    static void GetRowsX()
    {

        // Define ADO Objects.
        Recordset rstEmployees = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String line = null;
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        int intRows;
```

```

int intRecord;
int intUBound;
int intDisplaysize = 15;

try
{
    // Open recordset with names and hire dates from Employees
    rstEmployees = new Recordset();
    rstEmployees.open("SELECT fName, lName, hire_date " +
        "FROM Employee ORDER BY lName", strCnn,
        ADOEnums.CursorType.FORWARDONLY,
        ADOEnums.LockType.READONLY,
        ADOEnums.CommandType.TEXT);

    while(true)
    {
        // Get user input for number of rows.
        System.out.println(
            "\nEnter number of rows to retrieve. (<= 0 to Exit)");
        line = in.readLine().trim();

        // Convert string entry to int.
        intRows = Integer.parseInt(line);

        // Exit the application if intRows is negative or zero.
        if(intRows <= 0)
            break;

        // If GetRowsOK is successful, print the results,
        // noting if the end of the file was reached.
        if(GetRowsOK(rstEmployees,intRows))
        {
            SafeArray sa = avarRecords.toSafeArray();
            intUBound = sa.getUBound(2);
            if ( intRows > (intUBound + 1))
                System.out.println("\n(Not enough records in " +
                    "Recordset to retrieve " + intRows + " rows.)")
            System.out.println("\n" + (intUBound+ 1) +
                " records found.\n");

            // Print the retrieved data.

            for ( intRecord = sa.getLBound();
                intRecord <= intUBound; intRecord++)
            {

                System.out.println(
                    " " + sa.getString(0, intRecord) + " " +
                    sa.getString(1, intRecord) + ", " +
                    sa.getString(2, intRecord));
            }
        }
    }
}

```

```

        if ( ((intRecord +1) % intDisplaysize) == 0)
        {
            System.out.println("\nPress <Enter> to continue
            in.readLine();
        }
    }
}
else
{
    // Assuming the GetRows error was due to data
    // changes by another user, use Requery to
    // refresh the Recordset and start over.
    System.out.println("\nGetRows failed--retry? (Y/N)");
    if(in.readLine().trim().toUpperCase().equals("Y"))
        rstEmployees.requery();
    else
    {
        System.out.println("GetRows failed!");
        break;
    }
}

// Because using GetRows leaves the current
// record pointer at the last record accessed,
// move the pointer back to the beginning of the
// Recordset before looping back for another search.
rstEmployees.moveFirst();
}

// Cleanup objects before exit.
rstEmployees.close();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstEmployees != null)
    {
        PrintProviderError(rstEmployees.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
}

```

```

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

// Display Error that the application has attempted to convert
// a string of inappropriate format to one of the numeric type
catch(java.lang.NumberFormatException ne)
{
    System.out.println(
        "Exception: Must specify an Integer value." );
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
}
}

// GetRowsOK Function

static boolean GetRowsOK(Recordset rstTemp,int intNumber)
{
    // Store results of GetRows method in array.
    avarRecords = rstTemp.getRows(intNumber);

    // Return False only if fewer than the desired
    // number of rows were returned, but not because the
    // end of the Recordset was reached.
    if ( intNumber > (avarRecords.toSafeArray().getUBound(2)+ 1)
        && !(rstTemp.getEOF()))
        return false;
    else
        return true;
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
}

```

```

int i      = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndGetRowsJ

```

See Also

[BOF, EOF Properties](#) | [GetRows Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

GetString Method Example (VJ++)

This example demonstrates the [GetString](#) method.

Assume you are debugging a data access problem and want a quick, simple way of printing the current contents of a small [Recordset](#).

```
// BeginGetStringJ
// The WFC class includes the ADO objects.
import com.ms.wfc.data.*;
import java.io.* ;

public class GetStringX
{
    //The main entry point for the application.

    public static void main (String[] args)
    {
        GetStringX ();
        System.exit(0);
    }

    // GetStringX function
    static void GetStringX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstAuthors = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'";
        String strOutput;

        try
        {
            // Get the user input for state.
            System.out.println(
                "Enter a state (CA, IN, KS, MD, MI, OR, TN, UT): ");
            String strState = in.readLine().trim();
            String strQuery =
                "SELECT au_fname, au_lname, address, city FROM Authors "
                "WHERE state = '" + strState + "'";
        }
    }
}
```

```

// Open recordset with data from Authors table.
cnConn1 = new Connection();
cnConn1.open(strCnn);
rstAuthors = new Recordset();
rstAuthors.open(strQuery,
                cnConn1,
                AdoEnums.CursorType.STATIC,
                AdoEnums.LockType.READONLY,
                AdoEnums.CommandType.TEXT);

if (rstAuthors.getRecordCount() > 0)
{
    // Use all defaults: get all rows, TAB column delimiter,
    // CARRIAGE RETURN
    // row delimiter, empty-string null delimiter.
    strOutput =
        rstAuthors.getString(AdoEnums.StringFormat.CLIPSTRING
        rstAuthors.getRecordCount(),
        "\t ",
        "\n",
        "").trim();
    System.out.println("\nState = '" + strState + "'" +
        "\n\n" +
        "Name           Address           City" +
        "\n");
    System.out.println(strOutput);
}
else
    System.out.println("\nNo rows found for state = '" +
        strState + "'\n");

System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstAuthors != null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
}

```

```

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstAuthors != null)
        if (rstAuthors.getState() == 1)
            rstAuthors.close();
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i < nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{

```

```
        System.out.println("Error \n");
        System.out.println("\tSource = " + je.getClass() + "\n");
        System.out.println("\tDescription = " + je.getMessage() + "\n"
    }
}
// EndGetStringJ
```

See Also

[GetString Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

IsolationLevel and Mode Properties Example (VJ++)

This example uses the [Mode](#) property to open an exclusive connection, and the [IsolationLevel](#) property to open a transaction that is conducted in isolation of other transactions.

```
// BeginIsolationLevelJ
import com.ms.wfc.data.*;
import java.io.*;

public class IsolationLevelX
{
    // The main entry point for the application.
    public static void main (String[] args)
    {
        IsolationLevelX();
        System.exit(0);
    }

    // IsolationLevelX Function

    static void IsolationLevelX()
    {
        // Define ADO Objects
        Connection cnn1 = null;
        Recordset rstTitles = null;

        // Assign connection string to variable
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'

        // Declarations
        BufferedReader in =
            new BufferedReader ( new InputStreamReader(System.in));
        String line = null;

        try
        {
            // Open connection and Titles table
            cnn1 = new Connection();

            cnn1.setMode(AdoEnums.ConnectMode.SHAREEXCLUSIVE);
```

```

cnn1.setIsolationLevel(AdoEnums.IsolationLevel.ISOLATED);
cnn1.open(strCnn);

rstTitles = new Recordset();
rstTitles.setCursorType(AdoEnums.CursorType.DYNAMIC);
rstTitles.setLockType(AdoEnums.LockType.PESSIMISTIC);
rstTitles.open("Titles", cnn1, AdoEnums.CursorType.DYNAMIC,
    AdoEnums.LockType.PESSIMISTIC);

cnn1.beginTrans();

// Display the connection mode
if (cnn1.getMode() == AdoEnums.ConnectMode.SHAREEXCLUSIVE)
    System.out.println("\n\tConnection mode is exclusive");
else
    System.out.println("\n\tConnection mode is not exclusive
System.out.println("\nPress <Enter> to continue..");
in.readLine();

// Display the Isolation level
if (cnn1.getIsolationLevel() == AdoEnums.IsolationLevel.IS
    System.out.println("\tTransaction is Isolated\n");
else
    System.out.println("\tTransaction is not Isolated\n");
System.out.println("\nPress <Enter> to continue..");
in.readLine();

// Change the type of psychology titles
while(!rstTitles.getEOF())
{
    if(rstTitles.getField("Type").getString().trim().
        equals(new String("psychology")))
    {
        rstTitles.getField("Type").setString("self_help");
        rstTitles.update();
    }
    rstTitles.moveNext();
}

// Print current data in recordset
rstTitles.requery();
while(!rstTitles.getEOF())
{
    System.out.println(rstTitles.getField("Title").getString
        " - " + rstTitles.getField("Type").getString() );
    rstTitles.moveNext();
}
System.out.println("\nPress <Enter> to continue..");

```

```

        in.readLine();

        // Restore original data
        cnn1.rollbackTrans();
    }
    catch(AdoException ae)
    {
        // Notify the user of any errors that result from ADO

        // As passing a connection, check for null pointer first
        if(cnn1 !=null)
        {
            PrintProviderError(cnn1);
        }
        else
        {
            System.out.println("Exception:" + ae.getLocalizedMessage()
        }
    }

    // System Read requires this catch
    catch(java.io.IOException je)
    {
        PrintIOError(je);
    }

    finally
    {
        // Cleanup objects before exit.
        if (rstTitles != null)
            if (rstTitles.getState() == 1)
                rstTitles.close();
        if (cnn1 != null)
            if (cnn1.getState() == 1)
                cnn1.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

```

```

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndIsolationLevelJ

```

See Also

[IsolationLevel Property](#) | [Mode Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Item Property Example (VJ++)

This example demonstrates how the [Item](#) property accesses members of a collection. The example opens the *Authors* table of the *Pubs* database with a parameterized command.

The parameter in the command issued against the database is accessed from the [Command](#) object's [Parameters](#) collection by index and name. Then the fields of the returned [Recordset](#) are accessed from that object's [Fields](#) collection by index and name.

```
// BeginItemJ
import com.ms.wfc.data.*;
import java.io.* ;
import com.ms.com.*;

public class ItemX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        ItemX();
        System.exit(0);
    }

    // ItemX function

    static void ItemX()
    {

        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstAuthors = null;
        Command cmd = null;
        Parameter prm = null;
        Field fld = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'";
        Variant [] varColumn = null;
```

```

int intIndex;
int intLimit;

try
{
    cnConn1 = new Connection();
    rstAuthors = new Recordset();
    cmd = new Command();

    // Set the array with the Authors table field (column) names
    varColumn = new Variant[9];
    varColumn[0] = new Variant("au_id");
    varColumn[1] = new Variant("au_lname");
    varColumn[2] = new Variant("au_fname");
    varColumn[3] = new Variant("phone");
    varColumn[4] = new Variant("address");
    varColumn[5] = new Variant("city");
    varColumn[6] = new Variant("state");
    varColumn[7] = new Variant("zip");
    varColumn[8] = new Variant("contract");

    cmd.setCommandText("SELECT * FROM Authors WHERE state = ?")
    prm = cmd.createParameter("ItemXparm",
                              AdoEnums.DataType.CHAR,
                              AdoEnums.ParameterDirection.INPUT,
                              2,
                              "CA");
    cmd.getParameters().append(prm);

    cnConn1.open(strCnn);
    cmd.setActiveConnection(cnConn1);

    // Connection and CommandType are omitted
    // because a Command Object is provided.
    rstAuthors.open(cmd,
                    null,
                    AdoEnums.CursorType.STATIC,
                    AdoEnums.LockType.READONLY);

    System.out.println(
        "The Parameters collection accessed by index...");
    prm = cmd.getParameters().getItem(0);
    System.out.println("Parameter name = '" +
                       prm.getName() +
                       "', value = '" +
                       prm.getValue().toString() + "'\n");

    System.out.println(
        "The Parameters collection accessed by name...");
    prm = cmd.getParameters().getItem("ItemXparm");

```

```

System.out.println("Parameters name = '" +
                    prm.getName() +
                    "', value = '" +
                    prm.getValue().toString() + "'\n");
System.out.println("Press <Enter> to continue..");
in.readLine();

```

```

System.out.println(
    "The Fields collection accessed by index...");
rstAuthors.moveFirst();
intLimit = rstAuthors.getFields().getCount() - 1;
for(intIndex = 0; intIndex <= intLimit; intIndex++)
{
    fld = rstAuthors.getFields().getItem(intIndex);
    short intVtType = fld.getValue().getvt();
    String strFieldValue;
    switch(intVtType)
    {
        case Variant.VariantString :
            strFieldValue = fld.getValue().toString();
            break;
        case Variant.VariantBoolean :
            if(fld.getValue().getBoolean())
                strFieldValue = "True";
            else
                strFieldValue = "False";
            break;
        default :
            strFieldValue = fld.getValue().toString();
            break;
    }
    System.out.println("Field " +
                       Integer.toString(intIndex) +
                       " : Name = '" +
                       fld.getName() +
                       "', value = '" +
                       strFieldValue +
                       "'");
}
System.out.println("\nPress <Enter> to continue..");
in.readLine();

```

```

System.out.println("The Fields collection accessed by name.");
rstAuthors.moveFirst();
for(intIndex = 0; intIndex <= 8; intIndex++)
{
    fld = rstAuthors.getFields().getItem
        (varColumn[intIndex].toString());
    short intVtType = fld.getValue().getvt();

```

```

String strFieldValue;
switch(intVtType)
{
case Variant.VariantString :
    strFieldValue = fld.getValue().toString();
    break;
case Variant.VariantBoolean :
    if(fld.getValue().getBoolean())
        strFieldValue = "True";
    else
        strFieldValue = "False";
    break;
default :
    strFieldValue = fld.getValue().toString();
    break;
}
System.out.println("Field " +
                    "name = '" +
                    fld.getName() +
                    "', value = '" +
                    strFieldValue +
                    "'");
}
System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstAuthors != null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.

```

```

        if (rstAuthors != null)
            if (rstAuthors.getState() == 1)
                rstAuthors.close();
        if (cnConn1 != null)
            if (cnConn1.getState() == 1)
                cnConn1.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndItemJ

```

See Also

[Command Object](#) | [Fields Collection](#) | [Item Property](#) | [Parameters Collection](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

MarshalOptions Property Example (VJ++)

This example uses the [MarshalOptions](#) property to specify what rows are sent back to the server—All Rows or only Modified Rows.

```
// BeginMarshalOptionsJ
import java.io.*;
import com.ms.wfc.data.*;

public class MarshalOptionsX
{
    // The main entry point for the application.
    public static void main (String[] args)
    {
        MarshalOptionsX();
        System.exit(0);
    }
    // MarshalX Function

    static void MarshalOptionsX()
    {
        // Define ADO Objects
        Recordset rstEmployees = null;

        // Declarations
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        String line = null;

        try
        {
            // Open Recordset with names from Employees Table.
            String strCnn = " Provider='sqloledb';Data Source='MySqlSer
                "Initial Catalog='Pubs';Integrated Security='SSPI'";

            rstEmployees = new Recordset();
            rstEmployees.setCursorType(AdoEnums.CursorType.KEYSET);
            rstEmployees.setLockType(AdoEnums.LockType.OPTIMISTIC);
            rstEmployees.setCursorLocation(AdoEnums.CursorLocation.CLIE

            rstEmployees.open(
                "SELECT fname,lname from Employee ORDER BY lname",
```

```

        strConn, AdoEnums.CursorType.KEYSET,
        AdoEnums.LockType.OPTIMISTIC, AdoEnums.CommandType.TEXT)

// Store original data
String strOldFirst = rstEmployees.getField("fname").getString()
String strOldLast = rstEmployees.getField("lname").getString()

// Change data in edit buffer
rstEmployees.getField("fname").setString("Linda");
rstEmployees.getField("lname").setString("Kobara");

// Show contents of buffer and get user input
String strMessage = "Edit in progress: " + "\n" +
    "Original Data = \t" + strOldFirst + " " +
    strOldLast + "\n" + "Data in Buffer = \t" +
    rstEmployees.getField("fname").getString() + " " +
    rstEmployees.getField("lname").getString() + "\n" +
    "Use Update to replace the original data with " +
    "the buffered data in the recordset";
String strMarshalAll = "Would you like to send all the rows
    " in the recordset back to the server";
String strMarshalModified = "Would you like to send only " +
    " modified rows back to the server";

System.out.println(strMessage + "\nEnter (Y/N)...");

if (in.readLine().equalsIgnoreCase("Y"))
{
    System.out.println(strMarshalAll);
    System.out.println("\nEnter (Y/N)...");

    if(in.readLine().equalsIgnoreCase("Y"))
    {
        rstEmployees.setMarshalOptions(AdoEnums.MarshalOption
            rstEmployees.update());
    }
    else
    {
        System.out.println(strMarshalModified);
        System.out.println("\nEnter (Y/N)...");

        if (in.readLine().equalsIgnoreCase("Y"))
        {
            rstEmployees.setMarshalOptions(
                AdoEnums.MarshalOptions.MODIFIEDONLY);
            rstEmployees.update();
        }
    }
}
}

```

```

// Show the resulting data
System.out.println("\nData in recordset = " +
    rstEmployees.getField("fname").getString() +
    " " + rstEmployees.getField("lname").getString());

// Restore original data because this is a demonstration
if (!(strOldFirst.equals(rstEmployees.getField("fname"))
    &&(strOldLast.equals(rstEmployees.getField("lname")))))
{
    rstEmployees.getField("fname").setString(strOldFirst);
    rstEmployees.getField("lname").setString(strOldLast);
    rstEmployees.update();
}

System.out.println("\n\nPress <Enter> to continue..");
in.readLine();
}

catch(AdoException ae)
{
    // Notify the user of any errors that result from ADO

    // As passing a connection, check for null pointer first
    if(rstEmployees!= null)
    {
        PrintProviderError(rstEmployees.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getLocalizedMessage()
    }
}

// System Read requires this catch
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
}
}

```

```

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndMarshalOptionsJ

```

See Also

[MarshalOptions Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

MaxRecords Property Example (VJ++)

This example uses the [MaxRecords](#) property to open a [Recordset](#) containing the 10 most expensive titles in the *Titles* table.

```
// BeingMaxRecordsJ
import java.io.*;
import com.ms.wfc.data.*;

public class MaxRecordsX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        MaxRecordsX();
        System.exit(0);
    }
    // MaxRecordsX Function

    static void MaxRecordsX()
    {
        // Define ADO Objects
        Recordset rstTemp = null;

        try
        {
            // Declarations
            BufferedReader in =
                new BufferedReader(new InputStreamReader(System.in));

            // Open recordset containing the 10 most expensive
            // titles in the Titles table.
            String strCnn = " Provider='sqloledb';Data Source='MySqlSer
                " Initial Catalog='Pubs';Integrated Security='SSPI';";
            rstTemp = new Recordset();
            rstTemp.setMaxRecords(10);
            rstTemp.open("select title,price from Titles" +
                " order by price desc", strCnn,AdoEnums.CursorType.FORWA
                AdoEnums.LockType.READONLY, AdoEnums.CommandType.TEXT);

            // Display the contents of the recordset.
            System.out.println("Top Ten Titles by Price:\n");
        }
    }
}
```

```

while (!rstTemp.getEOF())
{
    System.out.println(" "+ rstTemp.getField("title").getStr
        " - " + rstTemp.getField("Price").getString());
    rstTemp.moveNext();
}

System.out.println("\n\nPress <Enter> to continue..");
in.readLine();
}

catch(AdoException ae)
{
    // Notify the user of any errors that result from ADO.

    // As passing a connection, check for null pointer first.
    if (rstTemp!=null)
    {
        PrintProviderError(rstTemp.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getLocalizedMessage
    }
}

// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstTemp != null)
        if (rstTemp.getState() == 1)
            rstTemp.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i      = 0;
}

```

```

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndMaxRecordsJ

```

See Also

[MaxRecords Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Move Method Example (VJ++)

This example uses the [Move](#) method to position the record pointer based on user input.

```
// BeginMoveJ
import java.io.*;
import com.ms.wfc.data.*;
import com.ms.com.*;

public class MoveX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        MoveX();
        System.exit(0);
    }
    // MoveX Function

    static void MoveX()
    {
        // Define ADO Objects
        Recordset rstAuthors = null;

        // Declarations
        String line = null;
        Variant varBookmark;
        String strCommand = null;
        int lngMove;
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        try
        {
            // Open recordset from Authors table.
            String strCnn = "Provider='sqloledb';Data Source='MySQLServ
                'Initial Catalog='Pubs';Integrated Security='SSPI'";

            rstAuthors = new Recordset();
            rstAuthors.setCursorType(AdoEnums.CursorType.STATIC);

            // Use client cursor to allow use of
            // Absolute Position property.
```

```

rstAuthors.setCursorLocation(AdoEnums.CursorLocation.CLIENT
rstAuthors.open("select au_id, au_fname, au_lname, city, state
    "from Authors order by au_lname",
        strCnn, AdoEnums.CursorType.STATIC,
        AdoEnums.CursorLocation.CLIENT, AdoEnums.CommandType.TEXT);

rstAuthors.moveFirst();

while(true)
{
    // Display information about current record and
    // ask how many records to move.
    strCommand = "Record:\t\t"+ rstAuthors.getAbsolutePosition()
        " of " + rstAuthors.getRecordCount() + "\n" + "\tAuthor
        + rstAuthors.getField("au_fname").getString() +
        " " + rstAuthors.getField("au_lname").getString() +
        "\n" + "\tLocation:\t" +
        rstAuthors.getField("city").getString() +
        ", " + rstAuthors.getField("state").getString()
        + "\n\n" + "\tEnter number of records to move" +
        " (positive or negative).";

    System.out.print("\t" + strCommand + "\t");
    line = in.readLine();

    // No entry exits program loop.
    if (line.length() == 0)
        break;

    // Converts string entry to int.
    lngMove = Integer.parseInt(line);

    // Store bookmark in case the move goes too far
    // forward or backward.
    varBookmark = (Variant)rstAuthors.getBookmark();

    // Move method requires parameter of data type int.
    rstAuthors.move(lngMove);

    // Trap for BOF and EOF.
    if (rstAuthors.getBOF())
    {
        System.out.println("\tToo far backward! " +
            "Returning to the current record.");
        rstAuthors.setBookmark(varBookmark);
    }
    if (rstAuthors.getEOF())
    {
        System.out.println("\tToo far forward! " +
            "Returning to the current record.");
    }
}

```

```

        rstAuthors.setBookmark(varBookmark);
    }
}
System.out.println("\tPress <Enter> to continue..");
in.readLine();
}

catch(ADOException ae)
{
    // Notify user of any errors that result from ADO.

    // As passing a recordset, check for null pointer.
    if (rstAuthors!=null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println(" Exception: "+ ae.getMessage());
    }
}
// System Read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}
// Required if the user enter non integer value.
catch(java.lang.NumberFormatException ne)
{
    System.out.println("\n\nPlease enter integer values!");
    rstAuthors.close();
}

finally
{
    // Cleanup objects before exit.
    if (rstAuthors != null)
        if (rstAuthors.getState() == 1)
            rstAuthors.close();
}
}
// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;

```

```

int i      = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndMoveJ

```

See Also

[Move Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

MoveFirst, MoveLast, MoveNext, and MovePrevious Methods Example (VJ++)

This example uses the [MoveFirst](#), [MoveLast](#), [MoveNext](#), and [MovePrevious](#) methods to move the record pointer of a [Recordset](#) based on the supplied command. The MoveAny procedure is required for this procedure to run.

```
// BeginMoveFirstJ
import com.ms.wfc.data.*;
import java.io.*;

public class MoveFirstX // DLL name.
{
    // Main Function

    public static void main( String rgArg[] )
    {
        MoveFirstX();
    }

    // MoveFirstX Function

    static void MoveFirstX()
    {
        // Declarations
        Recordset rsAuthors = null;
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        String line = null;

        String strCnn = "Provider='sqloledb';Data Source='MySQLServer' "
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        String strMessage = "UPDATE Titles SET type = 'psychology' "
            + "WHERE type = 'self_help'";

        int intCommand = 0;
        String strFName;
        String strLName;

        try
```

```

{
    // Open recordset from Authors table.
    rsAuthors = new Recordset();
    rsAuthors.setCursorLocation( AdoEnums.CursorLocation.CLIENT

    // Use client cursor to enable AbsolutePosition property.
    rsAuthors.open( "Authors", strCnn, AdoEnums.CursorType.STAT
        AdoEnums.LockType.BATCHOPTIMISTIC, AdoEnums.CommandType.

    // Get user's move requests and show current record informa
    while( true )    // Continuous loop.
    {
        // Assign field information to variable to simplify outp
        strFName = rsAuthors.getField("au_fname").getString();
        strLName = rsAuthors.getField("au_lname").getString();

        System.out.println
            ( "\nName: " + strFName + " " + strLName + "\n"
              + "Record " + rsAuthors.getAbsolutePosition()
              + " of " + rsAuthors.getRecordCount() + "\n\n" );
        System.out.println( "[1 - MoveFirst, 2 - MoveLast, \n");
        System.out.println( " 3 - MoveNext, 4 - MovePrevious]\n"

        // User types a number followed by enter (cr-lf).
        line = in.readLine();

        // No entry exits program loop.
        if (line.length() == 0) break;
        // Convert string entry to int.
        intCommand = Integer.parseInt(line);
        // Out of range entry exits program loop.
        if ((intCommand < 1) || (intCommand > 4)) break;

        // Call method based on user's validated selection.
        MoveAny(intCommand, rsAuthors);
    }
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rsAuthors != null)
    {
        PrintProviderError(rsAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

```

```

    }
    // System Read requires this catch.
    catch( java.io.IOException je )
    {
        PrintIOError(je);
    }

    finally
    {
        // Cleanup objects before exit.
        if (rsAuthors != null)
            if (rsAuthors.getState() == 1)
                rsAuthors.close();
    }
}

// MoveAny Function

static void MoveAny(int intChoice, Recordset rsTemp)
{
    // Move per selection from user, checking for BOF and EOF.
    try
    {
        switch(intChoice)
        {
            case 1: // Equals char of 1.
                rsTemp.moveFirst();
                break;
            case 2: // Equals char of 2.
                rsTemp.moveLast();
                break;
            case 3: // Equals char of 3.
                rsTemp.moveNext();
                if(rsTemp.getEOF())
                {
                    System.out.println("\nAlready at end of recordset!\n")
                    rsTemp.moveLast();
                }
                break;
            case 4: // Equals char of 4.
                rsTemp.movePrevious();
                if(rsTemp.getBOF())
                {
                    System.out.println
                    ("\nAlready at beginning of recordset!\n");
                    rsTemp.moveFirst();
                }
                break;
            default:

```

```

        break;
    }
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rsTemp != null)
    {
        PrintProviderError(rsTemp.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i < nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
}

```

```
        System.out.println("\tSource = " + je.getClass() + "\n");
        System.out.println("\tDescription = " + je.getMessage() + "\n"
    }
}
// EndMoveFirstJ
```

See Also

[MoveFirst, MoveLast, MoveNext, and MovePrevious Methods](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

NextRecordset Method Example (VJ++)

This example uses the [NextRecordset](#) method to view the data in a recordset that uses a compound command statement made up of three separate **SELECT** statements.

```
// BeginNextRecordsetJ
import java.io.*;
import com.ms.wfc.data.*;

public class NextRecordsetX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        NextRecordsetX();
        System.exit(0);
    }
    // NextRecordsetX Function
    static void NextRecordsetX()
    {
        // Define ADO Object
        Recordset rstCompound = null;

        // Declarations
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        String strCnn;
        int intCount;
        int intDisplayRecords = 15;
        int intRecordCount;

        try
        {
            // Open compound recordset.
            strCnn = "Provider='sqloledb';Data Source='MySQLServer';" +
                "Initial Catalog='Pubs';Integrated Security='SSPI';";

            rstCompound = new Recordset();
            rstCompound.open("select * from Authors;" +
                "select * from stores;" +
```

```

        "select * from jobs", strConn, AdoEnums.CursorType.FORWARD,
        AdoEnums.LockType.READONLY, AdoEnums.CommandType.TEXT);

// Display results from each select statement.
intCount=1;

while (rstCompound != null)
{
    System.out.println(
        "Contents of recordset #" + intCount + "\n");
    intRecordCount = 0;

    while(!rstCompound.getEOF())
    {
        System.out.println(
            rstCompound.getField(0).getString()+" " +
            rstCompound.getField(1).getString());
        intRecordCount++;

        rstCompound.moveNext();
        if ( intRecordCount == intDisplayRecords)
        {
            System.out.println("\nPress <Enter> to continue");
            in.readLine();
            intRecordCount = 0;
        }
    }
    System.out.println("\nPress <Enter> to continue..");
    in.readLine();

    rstCompound = rstCompound.nextRecordset();

    intCount++;
}
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}
catch(ADOException ae)
{
    // Notify the user of any errors that result from ADO.

    // As passing a recordset. check for the null pointer first
    if(rstCompound!=null)
    {
        PrintProviderError(rstCompound.getActiveConnection());
    }
    else

```

```

        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }
    catch(java.lang.NullPointerException ne)
    {
        System.out.println("Error Description: " + ne.getMessage())
    }

    finally
    {
        // Cleanup objects before exit.
        if (rstCompound != null)
            if (rstCompound.getState() == 1)
                rstCompound.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i < nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{

```

```
        System.out.println("Error \n");
        System.out.println("\tSource = " + je.getClass() + "\n");
        System.out.println("\tDescription = " + je.getMessage() + "\n"
    }
}
// EndNextRecordsetJ
```

See Also

[NextRecordset Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

NumericScale and Precision Properties Example (VJ++)

This example uses the [NumericScale](#) and [Precision](#) properties to display the numeric scale and precision of fields in the *Discounts* table of the *Pubs* database.

```
// BeginNumericScaleJ
import com.ms.wfc.data.*;
import java.io.* ;

public class NumericScaleX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        NumericScaleX();
        System.exit(0);
    }

    // NumericScaleX function

    static void NumericScaleX()
    {

        // Define ADO Objects.
        Recordset rstDiscounts = null;
        Field fldTemp = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";
        int intLoop;

        try
        {
            rstDiscounts = new Recordset();

            // Open recordset.
            rstDiscounts.open("Discounts", strCnn,
```

```

        AdoEnums.CursorType.FORWARDONLY,
        AdoEnums.LockType.READONLY,
        AdoEnums.CommandType.TABLE);

// Display numeric scale and precision of
// numeric and small integer fields.
for ( intLoop=0; intLoop <
    rstDiscounts.getFields().getCount();intLoop++)
{
    fldTemp = rstDiscounts.getFields().getItem(intLoop);
    if((fldTemp.getType()== AdoEnums.DataType.NUMERIC) |
        (fldTemp.getType()== AdoEnums.DataType.SMALLINT))
    {
        System.out.println("\nField: "
            + fldTemp.getName());
        System.out.println("\nNumeric scale: "
            + fldTemp.getNumericScale());
        System.out.println("\nPrecision: "
            + fldTemp.getPrecision());
        System.out.println("\n\nPress <Enter> to continue..")
            in.readLine();
    }
}

}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstDiscounts != null)
    {
        PrintProviderError(rstDiscounts.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.

```

```

        if (rstDiscounts != null)
            if (rstDiscounts.getState() == 1)
                rstDiscounts.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndNumericScaleJ

```

See Also

[NumericScale Property](#) | [Precision Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Open and Close Methods Example (VJ++)

This example uses the **Open** and [Close](#) methods on both [Recordset](#) and [Connection](#) objects that have been opened.

```
// BeginOpenJ
import com.ms.wfc.data.*;
import java.io.* ;
import com.ms.com.*;

public class OpenX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        OpenX();
        System.exit(0);
    }

    // OpenX function

    static void OpenX()
    {

        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstEmployees = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            + "Initial Catalog='Pubs';Integrated Security='SSP
        Variant varDate;
        String strHDate;

        try
        {
            // Open connection.
            cnConn1 = new Connection();
            cnConn1.open(strCnn);
```

```

// Open Employees table.
rstEmployees = new Recordset();
rstEmployees.setCursorType(AdoEnums.CursorType.KEYSET);
rstEmployees.setLockType(AdoEnums.LockType.OPTIMISTIC);
rstEmployees.open("Employee", cnConn1,
                 AdoEnums.CursorType.KEYSET,
                 AdoEnums.LockType.OPTIMISTIC,
                 AdoEnums.CommandType.TABLE);

// Assign the first employee record's hire date
// to a variable, then change the hire date.
varDate = rstEmployees.getField("hire_date").getOriginalVal
System.out.println("Original data\n");
System.out.println("\tName - Hire Date");
strHDate = rstEmployees.getField("hire_date").getString();
strHDate = strHDate.substring(5,7) + "/" +
           strHDate.substring(8,10)
           + "/" + strHDate.substring(2,4);
System.out.println("\t" +
                 rstEmployees.getField("fName").getString()+ " "
                 + rstEmployees.getField("lName").getString()+ " - "
                 + strHDate);
System.out.println("\nPress <Enter> to continue..");
in.readLine();
rstEmployees.getField("hire_date").setString("1/1/1900");
rstEmployees.update();
System.out.println("Changed data\n");
System.out.println("\tName - Hire Date");
strHDate = rstEmployees.getField("hire_date").getString();
strHDate = strHDate.substring(5,7) + "/" +
           strHDate.substring(8,10)
           + "/" + strHDate.substring(0,4);
System.out.println("\t" +
                 rstEmployees.getField("fName").getString()+ " "
                 + rstEmployees.getField("lName").getString()+ " - "
                 + strHDate);
System.out.println("\nPress <Enter> to continue..");
in.readLine();

// Requery Recordset and reset the hire date.
rstEmployees.requery();
rstEmployees.getField("hire_date").setValue(varDate);
rstEmployees.update();
System.out.println("Data after reset\n");
System.out.println("\tName - Hire Date");
strHDate = rstEmployees.getField("hire_date").getString();
strHDate = strHDate.substring(5,7) + "/" +
           strHDate.substring(8,10)
           + "/" + strHDate.substring(2,4);
System.out.println("\t" +

```

```

        rstEmployees.getField("fName").getString()+ " "
        + rstEmployees.getField("lName").getString()+ " - "
        + strHDate);
    System.out.println("\nPress <Enter> to continue..");
    in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstEmployees != null)
    {
        PrintProviderError(rstEmployees.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
    // Cleanup objects before exit.
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;

```

```

long nCount = 0;
int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndOpenJ

```

See Also

[Close Method](#) | [Connection Object](#) | [Open Method \(ADO Connection\)](#) | [Open Method \(ADO Recordset\)](#) | [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

OpenSchema Method Example (VJ++)

This example uses the [OpenSchema](#) method to display the name and type of each table in the *Pubs* database.

```
// BeginOpenSchemaJ
import com.ms.wfc.data.*;
import java.io.*;
import com.ms.com.*;

public class OpenSchemaX
{
    // The main entry point of the application.

public static void main (String[] args)
{
    System.out.println("\nResults for OpenSchemaX:\n\n");
    OpenSchemaX();
    System.out.println("\nResults for OpenSchemaX2:\n\n");
    OpenSchemaX2();
    System.exit(0);
}

// OpenSchemaX Function
static void OpenSchemaX()
{
    // Define ADO Objects
    Connection cnn1 = null;
    Recordset rstSchema = null;

    // Declarations
    String strCnn;
    BufferedReader in = new BufferedReader(new InputStreamReader(S
    int intDisplayRecords = 5;
    int intRecordCount = 0;

    try
    {

        cnn1 = new Connection();
        strCnn = "Provider = Microsoft.Jet.OLEDB.4.0;" +
            "Data Source=C:\\Program Files\\Microsoft " +
            "Office\\Office\\Samples\\Northwind.mdb;";
```

```

cnn1.open(strCnn);
rstSchema = cnn1.openSchema(AdoEnums.Schema.TABLES);

while (!rstSchema.getEOF())
{
    System.out.println("Table Name: " +
        rstSchema.getField("TABLE_NAME").getString()+"\n"+
        "Table Type: " +
        rstSchema.getField("TABLE_TYPE").getString()+"\n");
    intRecordCount++;
    if ( intRecordCount == intDisplayRecords)
    {
        System.out.println("Press <Enter> to continue..");
        in.readLine();
        intRecordCount = 0;
    }
    rstSchema.moveNext();
}
System.out.println("Press <Enter> to continue..");
in.readLine();
}
catch(AdoException ae)
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if(rstSchema != null)
    {
        PrintProviderError(rstSchema.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstSchema != null)
        if (rstSchema.getState() == 1)
            rstSchema.close();
}

```

```

        if (cnn1 != null)
            if (cnn1.getState() == 1)
                cnn1.close();
    }
}

// OpenSchemaX2 Function

static void OpenSchemaX2()
{
    // Define ADO Objects
    Connection cnn2 = null;
    Recordset rstSchema = null;

    // Declarations
    String strCnn;
    BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
    int intDisplayRecords = 5;
    int intRecordCount = 0;

    try
    {
        cnn2 = new Connection();
        strCnn = "Provider = Microsoft.Jet.OLEDB.4.0;" +
            "Data Source=C:\\Program Files\\Microsoft " +
            "Office\\Office\\Samples\\Northwind.mdb;";
        cnn2.open(strCnn);

        Variant[] va = new Variant[4];
        va[0] = new Variant();
        va[1] = new Variant();
        va[2] = new Variant();
        va[3] = new Variant("VIEW");
        rstSchema = cnn2.openSchema(AdoEnums.Schema.TABLES, (Object[]

    while (!rstSchema.getEOF())
    {
        System.out.println("Table Name: " +
            rstSchema.getField("TABLE_NAME").getString()+"\n"+
            "Table Type: " +
            rstSchema.getField("TABLE_TYPE").getString()+"\n");
        intRecordCount++;
        if ( intRecordCount == intDisplayRecords)
        {
            System.out.println("Press <Enter> to continue..");
            in.readLine();
            intRecordCount = 0;
        }
    }
}

```

```

        rstSchema.moveToNext();
    }
    System.out.println("Press <Enter> to continue..");
    in.readLine();
}
catch(AdoException ae)
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if(rstSchema != null)
    {
        PrintProviderError(rstSchema.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup Objects before exit.
    rstSchema.close();
    cnn2.close();
    // Cleanup objects before exit.
    if (rstSchema != null)
        if (rstSchema.getState() == 1)
            rstSchema.close();
    if (cnn2 != null)
        if (cnn2.getState() == 1)
            cnn2.close();
}

}
// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;

```

```

int i      = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndOpenSchemaJ

```

See Also

[OpenSchema Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Optimize Property Example (VJ++)

This example demonstrates the [Field](#) object [dynamic Optimize](#) property. The *zip* field of the *Authors* table in the *Pubs* database is not indexed. Setting the [Optimize](#) property to **True** on the *zip* field authorizes ADO to build an index that improves the performance of the [Find](#) method.

```
// BeginOptimizeJ
import com.ms.wfc.data.*;
import java.io.* ;

public class OptimizeX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        OptimizeX();
        System.exit(0);
    }

    // OptimizeX function

    static void OptimizeX()
    {

        // Define ADO Objects.
        Recordset rstAuthors = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";

        try
        {
            rstAuthors = new Recordset();
            rstAuthors.setCursorLocation(AdoEnums.CursorLocation.CLIENT
            // Enable index creation.
            rstAuthors.open("SELECT * FROM Authors",
                strCnn,
                AdoEnums.CursorType.STATIC,
                AdoEnums.LockType.READONLY,
                AdoEnums.CommandType.TEXT);
```

```

rstAuthors.getField("zip").getProperties().
    getItem("Optimize").setBoolean(true); // Create the index
rstAuthors.find("zip = '94595'"); // Find Akiko Yokomoto.
System.out.println(rstAuthors.getField("au_fname").getString() + " " +
    rstAuthors.getField("au_lname").getString() + " " +
    rstAuthors.getField("address").getString() + " " +
    rstAuthors.getField("city").getString() + " " +
    rstAuthors.getField("state").getString() + " ");
rstAuthors.getField("zip").getProperties().
    getItem("Optimize").setBoolean(false); // Delete the index

System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstAuthors != null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstAuthors != null)
        if (rstAuthors.getState() == 1)
            rstAuthors.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect

```

```

com.ms.wfc.data.Error ErrItem = null;
long nCount = 0;
int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndOptimizeJ

```

See Also

[Field Object | Optimize Property—Dynamic \(ADO\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

OriginalValue and UnderlyingValue Properties Example (VJ++)

This example demonstrates the [OriginalValue](#) and [UnderlyingValue](#) properties by displaying a message if a record's underlying data has changed during a [Recordset](#) batch update.

```
// BeginOriginalValueJ
import com.ms.wfc.data.*;
import java.io.* ;

public class OriginalValueX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        OriginalValueX();
        System.exit(0);
    }

    // OriginalValueX function

    static void OriginalValueX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstTitles = null;
        Field fldType = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));

        try
        {
            // Open a connection.
            cnConn1 = new Connection();
            cnConn1.open(strCnn);

            // Open recordset for batch update.
```

```

rstTitles = new Recordset();
rstTitles.setActiveConnection(cnConn1);
rstTitles.setCursorType(AdoEnums.CursorType.KEYSET);
rstTitles.setLockType(AdoEnums.LockType.BATCHOPTIMISTIC);
rstTitles.open("Titles", cnConn1,
               AdoEnums.CursorType.KEYSET,
               AdoEnums.LockType.BATCHOPTIMISTIC,
               AdoEnums.CommandType.TABLE);

// Set field object variable for Type field.
fldType = rstTitles.getField("type");

// Change the type of psychology titles.
while(!rstTitles.getEOF())
{
    if(rstTitles.getField("type").getString().
        trim().equals("psychology"))
        fldType.setString("self_help");
    rstTitles.moveNext();
}

// Similate a change by another user by updating
// data using a command string.
cnConn1.execute("UPDATE Titles SET type = 'sociology' "
               + "WHERE type = 'psychology'");

// Check for changes.
rstTitles.moveFirst();

while(!rstTitles.getEOF())
{
    String strOriginalValue =
        fldType.getOriginalValue().toString().trim();
    String strUnderlyingValue =
        fldType.getUnderlyingValue().toString().trim();
    if(!(strOriginalValue.equals(strUnderlyingValue)))
    {
        System.out.println("Data has changed!" + "\n\n");
        System.out.println("\tTitle ID: "
            + rstTitles.getField("title_id").getString().trim(
                "\n");
        System.out.println("\tCurrent value: "
            + fldType.getString()+ "\n");
        System.out.println("\tOriginal value: "
            + fldType.getOriginalValue().toString()+ "\n");
        System.out.println("\tUnderlying value: "
            + fldType.getUnderlyingValue().toString()+ "\n");
        System.out.println("\n\nPress <Enter> to continue..")
        in.readLine();
    }
}

```

```

        rstTitles.moveToNext();
    }
    // Cancel the update because this is a demonstration.

    rstTitles.cancelBatch();

    // Restore original values.
    cnConn1.execute("UPDATE Titles SET type = 'psychology' "
        + "WHERE type = 'sociology'");
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstTitles!= null)
    {
        PrintProviderError(rstTitles.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstTitles != null)
        if (rstTitles.getState() == 1)
            rstTitles.close();
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.

```

```

// ErrItem is an item object in the Connections Errors collect
com.ms.wfc.data.Error ErrItem = null;
long nCount = 0;
int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndOriginalValueJ

```

See Also

[OriginalValue Property](#) | [Recordset Object](#) | [UnderlyingValue Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Prepared Property Example (VJ++)

This example demonstrates the [Prepared](#) property by opening two [Command](#) objects—one prepared and one not prepared.

```
// BeginPreparedJ
import com.ms.wfc.data.*;
import java.io.* ;

public class PreparedX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        PreparedX();
        System.exit(0);
    }

    // PreparedX function

    static void PreparedX()
    {
        // Define string variables.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            + "Initial Catalog='Pubs';Integrated Security='SSPI'";
        String strCmd = "SELECT title, type FROM Titles ORDER BY type"

        // Define ADO Objects.
        Connection cnConn1 = null;
        Command cmd1 = null;
        Command cmd2 = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        long timeStart;
        long timeEnd;
        float timeNotPrepared ;
        float timePrepared;
        int  intLoop;
        String strTemp;

        try
        {
```

```

// Open a connection.
cnConn1 = new Connection();
cnConn1.open(strCnn);

// Create two command objects for the same
// command - one prepared and one not prepared.
cmd1 = new Command();
cmd1.setActiveConnection(cnConn1);
cmd1.setCommandType(AdoEnums.CommandType.TEXT);
cmd1.setCommandText(strCmd);

cmd2 = new Command();
cmd2.setActiveConnection(cnConn1);
cmd2.setCommandType(AdoEnums.CommandType.TEXT);
cmd2.setCommandText(strCmd);
cmd2.setPrepared(true);

// Set a timer, then execute the unprepared
// command 20 times.
timeStart = System.currentTimeMillis();
for ( intLoop = 0; intLoop < 20; intLoop++)
    cmd1.execute();
timeEnd = System.currentTimeMillis();
timeNotPrepared =(float)(timeEnd - timeStart)/1000f;

// Reset the timer, then execute the prepared
// command 20 times.
timeStart = System.currentTimeMillis();
for ( intLoop = 0; intLoop < 20; intLoop++)
    cmd2.execute();
timeEnd = System.currentTimeMillis();
timePrepared =(float)(timeEnd - timeStart)/1000f;

// Display performance results.
System.out.println("\nPerformance Results:");
System.out.println("\n\tNot Prepared: " + timeNotPrepared +
    " seconds");
System.out.println("\n\tPrepared: " + timePrepared +
    " seconds");
System.out.println("\n\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Connection, check for null pointer first.
    if (cnConn1!= null)
    {

```

```

        PrintProviderError(cnConn1);
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}
}

```

```
// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndPreparedJ
```

See Also

[Command Object](#) | [Prepared Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Provider and DefaultDatabase Properties Example (VJ++)

This example demonstrates the [Provider](#) property by opening three [Connection](#) objects using different providers. It also uses the [DefaultDatabase](#) property to set the default database for the Microsoft ODBC Provider.

```
// BeginProviderJ
import java.io.*;
import com.ms.wfc.data.*;

public class ProviderX
{
    // The main entry point of the application.
    public static void main (String[] args)
    {
        ProviderX();
        System.exit(0);
    }
    // ProviderX Function
    static void ProviderX()
    {
        // Define ADO Objects.
        Connection cnn1 = null;
        Connection cnn2 = null;
        Connection cnn3 = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        try
        {
            // Open a connection using the Microsoft ODBC Provider.
            cnn1 = new Connection();
            cnn1.setConnectionString("driver={SQL Server};"+
                "server='MySqlServer';User ID='MyUserID';Password='MyPa
            cnn1.open();
            cnn1.setDefaultDatabase("Pubs");

            // Display the provider.
            System.out.println("\n\n\tCnn1 provider: "+ cnn1.getProvide

            // Open connection using the OLE DB Provider for Microsoft
```

```

cnn2 = new Connection();
cnn2.setProvider("Microsoft.Jet.OLEDB.4.0");
cnn2.open("C:\\Program Files\\Microsoft Office\\Office\\Sam

// Display the provider.
System.out.println("\n\n\tCnn2 provider: " +
    cnn2.getProvider());

// Open a connection using the Microsoft SQL Server Provide
cnn3 = new Connection();
cnn3.setProvider("sqloledb");
cnn3.open("Data Source='MySqlServer';Initial Catalog='Pubs'

// Display the provider.
System.out.println("\n\n\tCnn3 provider: " +
    cnn3.getProvider());

System.out.println("\n\n\tPress <Enter> to continue..");
in.readLine();
}
catch(ADOException ae)
{
    // Notify the user of any errors that result from ADO.

    // As passing a Connection, check for null pointer first.
    if(cnn1 != null)
    {
        PrintProviderError(cnn1);
    }
    else
    {
        System.out.println("Exception: " + ae.getLocalizedMessage
    }
}
// System read requires needs this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (cnn1 != null)
        if (cnn1.getState() == 1)
            cnn1.close();
    if (cnn2 != null)
        if (cnn2.getState() == 1)
            cnn2.close();
    if (cnn3 != null)

```

```

        if (cnn3.getState() == 1)
            cnn3.close();
    }
}
// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndProviderJ

```

See Also

[Connection Object](#) | [DefaultDatabase Property](#) | [Provider Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Refresh Method Example (VJ++)

This example demonstrates using the [Refresh](#) method to refresh the [Parameters](#) collection for a stored procedure [Command](#) object.

```
// BeginRefreshJ
import com.ms.wfc.data.*;
import java.io.*;

public class RefreshX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        RefreshX();
        System.exit(0);
    }

    // RefreshX function

    static void RefreshX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Command cmdByRoyalty = null;
        Recordset rstByRoyalty = null;
        Recordset rstAuthors = null;

        //Declarations.
        String strAuthorID;
        String strFName;
        String strLName;
        int intRoyalty ;
        BufferedReader in =
            new BufferedReader (new InputStreamReader (System.in));
        String line = null;

        try
        {
            // Open a connection.
            String strCnn = "Provider='sqloledb';Data Source='MySQLServ
                + "Initial Catalog='Pubs';Integrated Security='

            cnConn1 = new Connection();
```

```

cnConn1.open(strCnn);

// Open a command object for a stored procedure
// with one parameter.
cmdByRoyalty = new Command();
cmdByRoyalty.setActiveConnection(cnConn1);
cmdByRoyalty.setCommandText("byRoyalty");
cmdByRoyalty.setCommandType(AdoEnums.CommandType.STOREDPROC);
cmdByRoyalty.getParameters().refresh();

// Get Parameter value and execute the command
// storing the results in the recordset.
System.out.println ("\nEnter Royalty : ");
line = in.readLine().trim();
intRoyalty = Integer.parseInt(line);
cmdByRoyalty.getParameters().getItem(1).setInt(intRoyalty);

// Create a recordset by executing the command.
rstByRoyalty = cmdByRoyalty.execute();

// Open the Authors table to get author names for display.
rstAuthors = new Recordset ();
rstAuthors.open(
    "Authors", strCnn, AdoEnums.CursorType.FORWARDONLY,
    AdoEnums.LockType.READONLY, AdoEnums.CommandType.TABLE );

// Print current data in the recordset,
// adding author names from Authors table.
System.out.println("\nAuthors with " + intRoyalty +
    " percent royalty");
while (!rstByRoyalty.getEOF())
{
    strAuthorID = rstByRoyalty.getField("au_id").getString();
    rstAuthors.setFilter("au_id ='" + strAuthorID + "'");
    strFName = rstAuthors.getField("au_fname").getString();
    strLName = rstAuthors.getField("au_lname").getString();
    System.out.println("\t" + strAuthorID + ", " + strFName
        + " " + strLName);
    rstByRoyalty.moveNext();
}
System.out.println("\n\nPress <Enter> key to continue..");
line = in.readLine();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstByRoyalty != null)

```

```

        {
            PrintProviderError(rstByRoyalty.getActiveConnection());
        }
        else if (rstAuthors != null)
        {
            PrintProviderError(rstAuthors.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }

    // This catch is required if input string cannot be converted
    // Integer data type.
    catch ( java.lang.NumberFormatException ne)
    {
        System.out.println("\nException: Integer Input required.
    }
    // System Read requires this catch.
    catch( java.io.IOException je )
    {
        PrintIOError(je);
    }

    finally
    {
        // Cleanup objects before exit.
        if (rstByRoyalty != null)
            if (rstByRoyalty.getState() == 1)
                rstByRoyalty.close();
        if (rstAuthors != null)
            if (rstAuthors.getState() == 1)
                rstAuthors.close();
        if (cnConn1 != null)
            if (cnConn1.getState() == 1)
                cnConn1.close();
    }
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i      = 0;

```

```

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndRefreshJ

```

See Also

[Command Object](#) | [Parameters Collection](#) | [Refresh Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Resync Method Example (VJ++)

This example demonstrates using the [Resync](#) method to refresh data in a static recordset.

```
// BeginResyncJ
import java.io.*;
import com.ms.wfc.data.*;

public class ResyncX
{
    // The main entry point of the application.
    public static void main (String[] args)
    {
        ResyncX();
        System.exit(0);
    }
    static void ResyncX()
    {
        // Define ADO Objects.
        Recordset rstTitles = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'

        try
        {
            // Open recordset for Titles table.
            rstTitles = new Recordset();
            rstTitles.setCursorLocation(AdoEnums.CursorLocation.CLIENT)
            rstTitles.setCursorType(AdoEnums.CursorType.STATIC);
            rstTitles.setLockType(AdoEnums.LockType.BATCHOPTIMISTIC);
            rstTitles.open("Titles", strCnn, AdoEnums.CursorType.STATIC
                AdoEnums.LockType.BATCHOPTIMISTIC,
                AdoEnums.CommandType.TABLE);

            // Change the type of the first title in the recordset.
            rstTitles.getField("type").setString("database");

            // Display the results of the change.
            System.out.println("\n\n\tBefore resync:\n" + "\tTitle - "
                rstTitles.getField("title").getString() +
```

```

        "\n\tType - " + rstTitles.getField("type").getString())

// Resync with database and redisplay the results.
rstTitles.resync();
System.out.println("\n\n\tAfter resync:\n" + "\tTitle - " +
    rstTitles.getField("title").getString() +
    "\n\tType - " +
    rstTitles.getField("type").getString()+"\n");
rstTitles.cancelBatch();

System.out.println("\tPress <Enter> to continue..");
in.readLine();

}
catch(AdoException ae)
{
    // Notify user of any errors that result from ADO.

    // As passing a recordset, check for null pointer first.
    if(rstTitles != null)
    {
        PrintProviderError(rstTitles.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstTitles != null)
        if (rstTitles.getState() == 1)
            rstTitles.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;

```

```

long nCount = 0;
int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i < nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndResyncJ

```

See Also

[Resync Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Save and Open Methods Example (VJ++)

These three examples demonstrate how the [Save](#) and **Open** methods can be used together.

Assume you are going on a business trip and want to take along a table from a database. Before you go, you access the data as a [Recordset](#) and save it in a transportable form. When you arrive at your destination, you access the **Recordset** as a local, disconnected **Recordset**. You make changes to the **Recordset**, then save it again, along with your changes. Finally, when you return home, you connect to the database again and update it with the changes you made on the road.

```
// BeginSaveJ
import com.ms.wfc.data.*;
import java.io.* ;

public class SaveX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        SaveX1();
        SaveX2();
        SaveX3();
        System.exit(0);
    }

    // First, access and save the Authors table.

    // SaveX1 function

    static void SaveX1()
    {
        // Define ADO Objects.
        Recordset rstAuthors = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
```

```

        "Initial Catalog='Pubs';Integrated Security='S
BufferedReader in =
    new BufferedReader(new InputStreamReader(System.in));
File file;

try
{
    rstAuthors = new Recordset();
    rstAuthors.setCursorLocation(AdoEnums.CursorLocation.CLIENT
    rstAuthors.open("SELECT * FROM Authors",
                    strCnn,
                    AdoEnums.CursorType.DYNAMIC,
                    AdoEnums.LockType.OPTIMISTIC,
                    AdoEnums.CommandType.TEXT);

    // For the sake of illustration, save the recordset to a
    //diskette in XML format.
    file = new File("c:\\Pubs.xml");
    if(!file.exists())
        rstAuthors.save("c:\\Pubs.xml",AdoEnums.PersistFormat.XML
    else
    {
        System.out.println("\nFile already exists.");
        System.out.println("\nPress <Enter> to continue..");
        in.readLine();
        System.exit(0);
    }
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstAuthors != null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{

```

```

        // Cleanup objects before exit.
        if (rstAuthors != null)
            if (rstAuthors.getState() == 1)
                rstAuthors.close();
    }
}

// At this point, you have arrived at your destination. You will
// access the Authors table as a local, disconnected Recordset.
// Don't forget you must have the MSPersist provider on the machi
// you are using in order to access the saved file, a:\Pubs.xml.

// SaveX2 function

static void SaveX2()
{
    // Define ADO Objects.
    Recordset rstAuthors = null;

    // Declarations.
    BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));

    try
    {
        rstAuthors = new Recordset();

        // For sake of illustration, we specify all parameters.
        rstAuthors.open("c:\\Pubs.xml",
            "Provider=MSPersist;",
            AdoEnums.CursorType.FORWARDONLY,
            AdoEnums.LockType.BATCHOPTIMISTIC,
            AdoEnums.CommandType.FILE);

        // Now you have a local, disconnected recordset.
        // Edit it as you desire.
        // (In this example, the change makes no difference).
        rstAuthors.find("au_lname = 'Carson'");
        if(rstAuthors.getEOF())
        {
            System.out.println("Name not found.");
            System.out.println("\nPress <Enter> to continue..");
            in.readLine();
            return;
        }
        rstAuthors.getField("city").setString("Berkeley");
        rstAuthors.update();

        // Save changes in ADTG format this time, for illustration.

```

```

        // Note that previous version on the diskette, as a:\Pubs.x
        rstAuthors.save("c:\\Pubs.adtg",AdoEnums.PersistFormat.ADTG
    }
    catch( AdoException ae )
    {
        // Notify user of any errors that result from ADO.

        // As passing a Recordset, check for null pointer first.
        if (rstAuthors != null)
        {
            PrintProviderError(rstAuthors.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }

    // System read requires this catch.
    catch( java.io.IOException je)
    {
        PrintIOError(je);
    }

    finally
    {
        // Cleanup objects before exit.
        if (rstAuthors != null)
            if (rstAuthors.getState() == 1)
                rstAuthors.close();
    }
}

// Finally, update the database with your changes.

// SaveX3 function

static void SaveX3()
{
    // Define ADO Objects.
    Connection cnConn1 = null;
    Recordset rstAuthors = null;

    // Declarations.
    String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
                    "Initial Catalog='Pubs';Integrated Security='S

    try
    {

```

```

// If there is no ActiveConnection, you can open with default
rstAuthors = new Recordset();
rstAuthors.open("c:\\Pubs.adtg");

// Connect to the database, associate the Recordset with
// connection, then update the database table with the changes
// Recordset.
cnConn1 = new Connection();
cnConn1.open(strCnn);
rstAuthors.setActiveConnection(cnConn1);
rstAuthors.updateBatch();
}
catch( AdoException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstAuthors != null)
    {
        PrintProviderError(rstAuthors.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

finally
{
    // Cleanup objects before exit.
    if (rstAuthors != null)
        if (rstAuthors.getState() == 1)
            rstAuthors.close();
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collection
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;

```

```

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndSaveJ

```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Sort Property Example (VJ++)

This example uses the [Recordset](#) object's [Sort](#) property to reorder the rows of a **Recordset** derived from the *Authors* table of the *Pubs* database. A secondary utility routine prints each row.

```
// BeginSortJ
import com.ms.wfc.data.*;
import java.io.* ;

public class SortX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        SortX();
        System.exit(0);
    }

    // SortX function

    static void SortX()
    {
        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstAuthors = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
                        "Initial Catalog='Pubs';Integrated Security='S

    try
    {
        cnConn1 = new Connection();
        cnConn1.open(strCnn);
        rstAuthors = new Recordset();
        rstAuthors.setCursorLocation(AdoEnums.CursorLocation.CLIENT
        rstAuthors.open("SELECT * FROM Authors",
                        cnConn1,
                        AdoEnums.CursorType.STATIC,
                        AdoEnums.LockType.READONLY,
                        AdoEnums.CommandType.TEXT);
        SortXprint("Initial Order",rstAuthors);
    }
}
```

```

        rstAuthors.setSort("au_lname ASC, au_fname ASC");
        SortXprint("Last Name Ascending",rstAuthors);

        rstAuthors.setSort("au_lname DESC, au_fname ASC");
        SortXprint("Last Name Descending",rstAuthors);
    }
    catch( AdoException ae )
    {
        // Notify user of any errors that result from ADO.

        // As passing a Recordset, check for null pointer first.
        if (rstAuthors != null)
        {
            PrintProviderError(rstAuthors.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }

    finally
    {
        // Cleanup objects before exit.
        if (rstAuthors != null)
            if (rstAuthors.getState() == 1)
                rstAuthors.close();
        if (cnConn1 != null)
            if (cnConn1.getState() == 1)
                cnConn1.close();
    }
}

// SortXprint function

static void SortXprint(String strTitle,Recordset rstp)
{
    // Declarations.
    BufferedReader in =
        new BufferedReader (new InputStreamReader(System.in));
    int intDisplaysize = 15;
    int intCount = 1;
    try
    {
        System.out.println("-----" +
            strTitle +
            "-----");
        System.out.println("First Name    Last Name" + "\n" +
            "-----" +

```

```

        "-----");
rstp.moveFirst();
while(!rstp.getEOF())
{
    System.out.println(rstp.getField("au_fname").getString()
        " " +
        rstp.getField("au_lname").getString());
    if(intCount % intDisplaysize == 0)
    {
        System.out.println("\nPress <Enter> to continue..");
        in.readLine();
        intCount = 0;
    }
    intCount++;
    rstp.moveNext();
}
System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( ADOException ae )
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstp != null)
    {
        PrintProviderError(rstp.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}
}
// PrintProviderError Function

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i = 0;
}

```

```

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if( nCount > 0);
{
    // Collection ranges from 0 to nCount - 1
    for (i = 0; i< nCount; i++)
    {
        ErrItem = Cnn1.getErrors().getItem(i);
        System.out.println("\t Error number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription() );
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndSortJ

```

See Also

[Recordset Object](#) | [Sort Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Source Property Example (VJ++)

This example demonstrates the [Source](#) property by opening three [Recordset](#) objects based on different data sources.

```
// BeginSourceJ
import java.io.*;
import com.ms.wfc.data.*;

public class SourceX
{
    // The main entry point of the application.
    public static void main (String[] args)
    {
        SourceX();
        System.exit(0);
    }
    // SourceX Function
    static void SourceX()
    {
        // Define string variables.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'
        String strSQL = "SELECT title_ID AS TitleID, title AS Title,"+
            "publishers.pub_id AS PubID, pub_name AS PubName "+
            "FROM publishers INNER JOIN Titles "+
            "ON publishers.pub_id=Titles.pub_id "+
            "ORDER BY Title";

        // Define ADO Objects.
        Connection cnn1 = null;
        Recordset rstTitles = null;
        Recordset rstPublishers = null;
        Recordset rstPublishersDirect = null;
        Recordset rstTitlesPublishers = null;
        Command cmdSQL = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        try
        {
            // Open a connection.
            cnn1 = new Connection();
```

```

cnn1.open(strCnn);

// Open a recordset based on a command object.
cmdSQL = new Command();
cmdSQL.setActiveConnection(cnn1);
cmdSQL.setCommandText("Select title,type,pubdate " +
    "FROM Titles ORDER BY title");

rstTitles = new Recordset();
rstTitles = cmdSQL.execute();

// Open a recordset based on a table.
rstPublishers = new Recordset();
rstPublishers.open("publishers", strCnn,
    AdoEnums.CursorType.FORWARDONLY, AdoEnums.LockType.READO
    AdoEnums.CommandType.TABLE);

// Open a recordset based on a table.
rstPublishersDirect = new Recordset();
rstPublishersDirect.open("publishers", strCnn,
    AdoEnums.CursorType.FORWARDONLY, AdoEnums.LockType.READO
    AdoEnums.CommandType.TABLEDIRECT);

// Open a recordset based on an SQL String.
rstTitlesPublishers = new Recordset();

rstTitlesPublishers.open(strSQL, strCnn,
    AdoEnums.CursorType.FORWARDONLY, AdoEnums.LockType.READO
    AdoEnums.CommandType.TEXT);

// Use Source property to display the source of each record
System.out.println("\nrstTitles source: \n"+
    rstTitles.getSource());
System.out.println("\nrstPublishers source: \n"+
    rstPublishers.getSource());
System.out.println("\nrstPublishersDirect source:\n" +
    rstPublishersDirect.getSource() );
System.out.println("\nrstTitlesPublishers source: \n" +
    rstTitlesPublishers.getSource());

System.out.println("\n\nPress <Enter> to continue..");
in.readLine();
}
catch(ADOException ae)
{
    // Notify the user of any errors that result from ADO.

    // As passing a recordset, check for null pointer first.

    if(rstPublishers != null)

```

```

    {
        PrintProviderError(rstPublishers.getActiveConnection());
    }
    else if(rstPublishersDirect != null)
    {
        PrintProviderError(
            rstPublishersDirect.getActiveConnection());
    }
    else if(rstTitles != null)
    {
        PrintProviderError(rstTitles.getActiveConnection());
    }
    else if(rstTitlesPublishers != null)
    {
        PrintProviderError(
            rstTitlesPublishers.getActiveConnection());
    }
    else
        System.out.println("Exception: " + ae.getMessage());
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}
finally
{
    // Cleanup objects before exit.
    if (rstTitles != null)
        if (rstTitles.getState() == 1)
            rstTitles.close();
    if (rstPublishers != null)
        if (rstPublishers.getState() == 1)
            rstPublishers.close();
    if (rstPublishersDirect != null)
        if (rstPublishersDirect.getState() == 1)
            rstPublishersDirect.close();
    if (rstTitlesPublishers != null)
        if (rstTitlesPublishers.getState() == 1)
            rstTitlesPublishers.close();
    if (cnn1 != null)
        if (cnn1.getState() == 1)
            cnn1.close();
}
}
// PrintProviderError Function

```

```

static void PrintProviderError( Connection Cnn1 )
{
    // Print Provider errors from Connection object.
    // ErrItem is an item object in the Connections Errors collect
    com.ms.wfc.data.Error ErrItem = null;
    long nCount = 0;
    int i      = 0;

    nCount = Cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if( nCount > 0);
    {
        // Collection ranges from 0 to nCount - 1
        for (i = 0; i< nCount; i++)
        {
            ErrItem = Cnn1.getErrors().getItem(i);
            System.out.println("\t Error number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription() );
        }
    }
}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)
{
    System.out.println("Error \n");
    System.out.println("\tSource = " + je.getClass() + "\n");
    System.out.println("\tDescription = " + je.getMessage() + "\n"
}
}
// EndSourceJ

```

See Also

[Recordset Object](#) | [Source Property \(ADO Recordset\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

State Property Example (VJ++)

This example uses the [State](#) property to display a message while [asynchronous](#) connections are opening and asynchronous commands are executing.

```
// BeginStateJ
import com.ms.wfc.data.*;
import java.io.*;

public class StateX
{
    // The main entry point of the application.
    public static void main (String[] args)
    {
        StateX();
        System.exit(0);
    }
    // StateX Function
    static void StateX()
    {
        // Define ADO Objects.
        Connection cnn1 = null;
        Connection cnn2 = null;
        Command cmdChange = null;
        Command cmdRestore = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        try
        {
            // Open two Asynchronous connections, displaying
            // a message while connecting.
            cnn1 = new Connection();
            cnn2 = new Connection();

            cnn1.open(strCnn, "", "", AdoEnums.ConnectOption.ASYNCCONNECT)
            while(cnn1.getState()==AdoEnums.ObjectState.CONNECTING)
                System.out.println("Opening the first connection....");

            cnn2.open(strCnn, "", "", AdoEnums.ConnectOption.ASYNCCONNECT)
            while(cnn2.getState()==AdoEnums.ObjectState.CONNECTING)
```

```

        System.out.println("Opening the second connection....");

// Create two command Objects.
cmdChange = new Command();
cmdChange.setActiveConnection(cnn1);
cmdChange.setCommandText("UPDATE Titles SET type = 'self_he
                           "WHERE type = 'psychology'");

cmdRestore = new Command();
cmdRestore.setActiveConnection(cnn2);
cmdRestore.setCommandText(
    "UPDATE Titles SET type = 'psychology'"+
    "WHERE type = 'self_help'");

// Executing the commands, displaying a message
// while they are executing.
cmdChange.execute(null, AdoEnums.ExecuteOption.ASYNCEXECUTE)

while(cmdChange.getState() == AdoEnums.ObjectState.EXECUTIN
    System.out.println("Change command executing....");

cmdRestore.execute(null, AdoEnums.ExecuteOption.ASYNCEXECUTE

while(cmdRestore.getState() == AdoEnums.ObjectState.EXECUTI
    System.out.println("Restore command executing....");

System.out.println("Press <Enter> to continue..");
in.readLine();
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}
catch(ADOException ae)
{
    // Notify user of any errors resulting from ADO.

// As passing a connection, check for null pointer first.
if(cnn1 != null)
{
    System.out.println("The error has occurred in cnn1:");
    PrintProviderError(cnn1);
}
else if(cnn2 != null)
{
    System.out.println("The error has occurred in cnn2:");
    PrintProviderError(cnn2);
}
else

```

```

        {
            System.out.println("Exception: "+ ae.getMessage());
        }
    }

    finally
    {
        // Cleanup objects before exit.
        if (cnn1 != null)
            if (cnn1.getState() == 1)
                cnn1.close();
        if (cnn2 != null)
            if (cnn2.getState() == 1)
                cnn2.close();
    }
}

// PrintProviderError Function
static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Collect
    com.ms.wfc.data.Error          ErrItem = null;
    long                            nCount = 0;
    int                              i = 0;

    nCount = cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if ( nCount > 0)
    {
        // Collection ranges from 0 to nCount-1.
        for ( i=0;i<nCount; i++)
        {
            ErrItem = cnn1.getErrors().getItem(i);
            System.out.println("\t Error Number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription());
        }
    }
}

// PrintIOError Function
static void PrintIOError(java.io.IOException je)
{
    System.out.println("Error: \n");
    System.out.println("\t Source: " + je.getClass() + "\n");
    System.out.println("\t Description: "+ je.getMessage() + "\n")
}

}
// EndStateJ

```

See Also

[State Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Status Property Example (VJ++)

This example uses the [Status](#) property to display which records have been modified in a batch operation before a batch update has occurred.

```
// BeginStatusJ
import java.io.*;
import com.ms.wfc.data.*;

public class StatusX
{
    // The main entry point of the application.

    public static void main (String[] args)
    {
        StatusX();
        System.exit(0);
    }
    // StatusX Function

    static void StatusX()
    {
        // Define ADO Objects.
        Recordset rstTitles = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySQLServer'
                        'Initial Catalog='Pubs';Integrated Security='SSPI'";
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        try
        {
            // Open Recordset for batch update.
            rstTitles = new Recordset();
            rstTitles.setCursorType(AdoEnums.CursorType.KEYSET);
            rstTitles.setLockType(AdoEnums.LockType.BATCHOPTIMISTIC);
            rstTitles.open("Titles", strCnn, AdoEnums.CursorType.KEYSET
                AdoEnums.LockType.BATCHOPTIMISTIC,
                AdoEnums.CommandType.TABLE);

            // Change the type of psychology titles.
            while(!rstTitles.getEOF())
            {
                if(rstTitles.getField("Type").getString().trim().
```

```

        equals(new String("psychology")))
        rstTitles.getField("Type").setString("self_help");

    rstTitles.moveNext();
}

// Display Title ID and status.
rstTitles.moveFirst();

while(!rstTitles.getEOF())
{
    if(rstTitles.getStatus()==AdoEnums.RecordStatus.MODIFIED
        System.out.println(rstTitles.getField("title_id").
            getString() + "- Modified");
    else
        System.out.println(rstTitles.getField("title_id").
            getString());
    rstTitles.moveNext();
}

// Cancel the update because this is a demonstration.
rstTitles.cancelBatch();

System.out.println("Press <Enter> to continue..");
in.readLine();
}
catch(AdoException ae)
{
    // Notify the user of any errors that result from ADO.

    // As passing a Recordset, check for the null pointer first
    if(rstTitles != null)
    {
        PrintProviderError(rstTitles.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstTitles != null)

```

```

        if (rstTitles.getState() == 1)
            rstTitles.close();
    }
}
// PrintProviderError Function
static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Collect
    com.ms.wfc.data.Error          ErrItem = null;
    long                            nCount = 0;
    int                              i = 0;

    nCount = cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if ( nCount > 0)
    {
        // Collection ranges from 0 to nCount-1.
        for ( i=0;i<nCount; i++)
        {
            ErrItem = cnn1.getErrors().getItem(i);
            System.out.println("\t Error Number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription());
        }
    }
}
// PrintIOError Function
static void PrintIOError(java.io.IOException je)
{
    System.out.println("Error: \n");
    System.out.println("\t Source: " + je.getClass() + "\n");
    System.out.println("\t Description: "+ je.getMessage() + "\n")
}
}
// EndStatusJ

```

See Also

[Status Property \(ADO Recordset\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

StayInSync Property Example (VJ++)

This example demonstrates how the [StayInSync](#) property facilitates accessing rows in a hierarchical [Recordset](#).

The outer loop displays each author's first and last name, state, and identification. The appended **Recordset** for each row is retrieved from the [Fields](#) collection and automatically assigned to **rstTitleAuthor** by the **StayInSync** property whenever the parent **Recordset** moves to a new row. The inner loop displays four fields from each row in the appended recordset.

```
// BeginStayInSyncJ
import com.ms.wfc.data.*;
import java.io.* ;
import com.ms.com.*;

public class StayInSyncX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        StayInSyncX();
        System.exit(0);
    }

    // StayInSyncX function

    static void StayInSyncX()
    {

        // Define ADO Objects.
        Connection cnConn1 = null;
        Recordset rstAuthors = null;
        Recordset rstTitleAuthor = null;

        // Declarations.
        BufferedReader in = new
            BufferedReader (new InputStreamReader(System.in));
        String strCnn = "Provider=MSDataShape;" +
            "Data Provider='sqloledb';Data Source='MySqlServer';" +
```

```

"Initial Catalog='Pubs';Integrated Security='SSPI';";

try
{
    cnConn1 = new Connection();
    cnConn1.open(strCnn);
    rstAuthors = new Recordset();
    rstAuthors.setStayInSync(true);
    rstAuthors.open("SHAPE {select * from Authors} " +
        "APPEND ({select * from titleauthor}" +
        "RELATE au_id TO au_id) AS chapTitleAuthor",
        cnConn1,
        AdoEnums.CursorType.STATIC,
        AdoEnums.LockType.READONLY,
        AdoEnums.CommandType.TEXT);

    Variant varRstTitleAuthor = rstAuthors.getFields().
        getItem("chapTitleAuthor").getValue();
    rstTitleAuthor =new Recordset(varRstTitleAuthor.toObject())
    int intCount =0;
    while(!rstAuthors.getEOF())
    {
        System.out.println("\n" +
            rstAuthors.getField("au_fname").getString() + " " +
            rstAuthors.getField("au_lname").getString() + " " +
            rstAuthors.getField("state").getString() + " " +
            rstAuthors.getField("au_id").getString());
        while(!rstTitleAuthor.getEOF())
        {
            System.out.println(rstTitleAuthor.getField(0).
                getString() + " " +
                rstTitleAuthor.getField(1).getString() + " " +
                rstTitleAuthor.getField(2).getString() + " " +
                rstTitleAuthor.getField(3).getString());
            rstTitleAuthor.moveNext();
        }
        if(++intCount % 5 == 0)
        {
            System.out.println("\nPress <Enter> to continue..");
            in.readLine();
        }
        rstAuthors.moveNext();
    }

    System.out.println("\nPress <Enter> to continue..");
    in.readLine();
}
catch( AdoException ae )
{

```

```

// Notify user of any errors that result from ADO.

// As passing a Recordset, check for null pointer first.
if (rstAuthors != null)
{
    PrintProviderError(rstAuthors.getActiveConnection());
}
else
{
    System.out.println("Exception: " + ae.getMessage());
}
}

// System read requires this catch.
catch( java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstTitleAuthor != null)
        if (rstTitleAuthor.getState() == 1)
            rstTitleAuthor.close();
    if (rstAuthors != null)
        if (rstAuthors.getState() == 1)
            rstAuthors.close();
    if (cnConn1 != null)
        if (cnConn1.getState() == 1)
            cnConn1.close();
}
}

// PrintProviderError Function
static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Collect
    com.ms.wfc.data.Error          ErrItem = null;
    long                            nCount = 0;
    int                              i = 0;

    nCount = cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if ( nCount > 0)
    {
        // Collection ranges from 0 to nCount-1.

```

```
        for ( i=0;i<nCount; i++)
        {
            ErrItem = cnn1.getErrors().getItem(i);
            System.out.println("\t Error Number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription());
        }
    }
}
// PrintIOError Function
static void PrintIOError(java.io.IOException je)
{
    System.out.println("Error: \n");
    System.out.println("\t Source: " + je.getClass() + "\n");
    System.out.println("\t Description: "+ je.getMessage() + "\n")
}
}
// EndStayInSyncJ
```

See Also

[Fields Collection](#) | [Recordset Object](#) | [StayInSync Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Supports Method Example (VJ++)

This example uses the [Supports](#) method to display the options supported by a recordset opened with different [cursor](#) types. The DisplaySupport function is required for this example to run.

```
// BeginSupportsJ
import com.ms.wfc.data.*;
import java.io.*;

public class SupportsX
{
    // The main entry point of the application.

    public static void main (String[] args)
    {
        SupportsX();
        System.exit(0);
    }
    // SupportsX Function

    static void SupportsX()
    {
        // Define ADO Objects.
        Recordset rstTitles = null;

        // Declarations.
        int[] aintCursorType = new int[4];
        String strCnn;
        int intIndex;
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        try
        {
            // Open connections.
            strCnn = "Provider='sqloledb';Data Source='MySqlServer';"+
                "Initial Catalog='Pubs';Integrated Security='SSPI'";

            // Fill array with CursorType constants.
            aintCursorType[0] = AdoEnums.CursorType.FORWARDONLY;
            aintCursorType[1] = AdoEnums.CursorType.KEYSET;
            aintCursorType[2] = AdoEnums.CursorType.DYNAMIC;
            aintCursorType[3] = AdoEnums.CursorType.STATIC;
```

```

// Open recordset using each CursorType and
// Optimistic Locking. Then call the DisplaySupport
// procedure to display the supported options.
for(intIndex = 0; intIndex < 4; intIndex++)
{
    rstTitles = new Recordset();
    rstTitles.setCursorType(aintCursorType[intIndex]);
    rstTitles.setLockType(AdoEnums.LockType.OPTIMISTIC);
    rstTitles.open("Titles", strCnn, aintCursorType[intIndex]
        AdoEnums.LockType.OPTIMISTIC, AdoEnums.CommandType.TA

    switch(aintCursorType[intIndex])
    {
    case AdoEnums.CursorType.FORWARDONLY:
        System.out.println("ForwardOnly cursor supports:");
        break;
    case AdoEnums.CursorType.KEYSET:
        System.out.println("Keyset cursor supports:");
        break;
    case AdoEnums.CursorType.DYNAMIC:
        System.out.println("Dynamic cursor supports:");
        break;
    case AdoEnums.CursorType.STATIC:
        System.out.println("Static cursor supports:");
        break;
    default:
        break;
    }
    DisplaySupport(rstTitles);
    System.out.println("Press <Enter> to continue..");
    in.readLine();
}
}
catch(AdoException ae)
{
    // Notify user of any errors that result from ADO.

    // As passing a Recordset, check for null pointer first.
    if (rstTitles!= null)
    {
        PrintProviderError(rstTitles.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System read requires this catch.
catch( java.io.IOException je)
{

```



```

// Print Provider Errors from Connection Object.
// ErrItem is an item object in the Connections Errors Collect
com.ms.wfc.data.Error          ErrItem = null;
long                          nCount = 0;
int                            i = 0;

nCount = cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.
if ( nCount > 0)
{
    // Collection ranges from 0 to nCount-1.
    for ( i=0;i<nCount; i++)
    {
        ErrItem = cnn1.getErrors().getItem(i);
        System.out.println("\t Error Number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription());
    }
}
}
// PrintIOError Function
static void PrintIOError(java.io.IOException je)
{
    System.out.println("Error: \n");
    System.out.println("\t Source: " + je.getClass() + "\n");
    System.out.println("\t Description: "+ je.getMessage() + "\n")
}
}
// EndSupportsJ

```

See Also

[Recordset Object](#) | [Supports Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Type Property Example (Field) (VJ++)

This example demonstrates the [Type](#) property by displaying the name of the constant that corresponds to the value of the **Type** property of all the [Field](#) objects in the *Employees* table. The `FieldType` function is required for this procedure to run.

```
// BeginFieldTypeJ
import java.io.*;
import com.ms.wfc.data.*;

public class TypeFieldX
{
    // The main entry point of the application.

    public static void main (String[] args)
    {
        TypeFieldX();
        System.exit(0);
    }

    // TypeFieldX Function
    static void TypeFieldX()
    {
        // Define ADO Objects.
        Recordset rstEmployees = null;
        Field fldLoop = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'";
        int intLoop;
        int intRecordCount = 0;
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        try
        {
            // Open the Recordset with data from Employees table.
            rstEmployees = new Recordset();
            rstEmployees.open("employee", strCnn,
```

```

        AdoEnums.CursorType.FORWARDONLY, AdoEnums.LockType.READC
        AdoEnums.CommandType.TABLE);

System.out.println("Fields in the Employees table:\n");

// Enumerate fields collection of Employees table.
for(intLoop = 0;intLoop <
    rstEmployees.getFields().getCount();intLoop++)
{
    intRecordCount++;
    // Loop needed for display of records
    if((intRecordCount % 6)==0)
    {
        System.out.println("Press <Enter> to continue..");
        in.readLine();
    }

    fldLoop = rstEmployees.getFields().getItem(intLoop);
    System.out.println("  Name:" + fldLoop.getName() + "\n"+
        "  Type:" + FieldType(fldLoop.getType()) + "\n");

}
System.out.println("Press <Enter> to continue");
in.readLine();
}
catch(AdoException ae)
{
    // Notify the user of any errors that result from ADO.

    // As passing a Recordset, check for the null pointer first
    if(rstEmployees != null)
    {
        PrintProviderError(rstEmployees.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)

```

```

        rstEmployees.close();
    }
}
// FieldType Function
static String FieldType( int intType )
{
    String strLoop = null;

    switch(intType)
    {
    case AdoEnums.DataType.CHAR:
        strLoop = "adChar";
        break;
    case AdoEnums.DataType.VARCHAR:
        strLoop ="adVarChar";
        break;
    case AdoEnums.DataType.SMALLINT:
        strLoop = "adSmallInt";
        break;
    case AdoEnums.DataType.UNSIGNEDTINYINT:
        strLoop = "adUnsignedTinyInt" ;
        break;
    case AdoEnums.DataType.DBTIMESTAMP:
        strLoop = "adDBTimeStamp";
        break;
    default:
        break;
    }

    return strLoop;
}

// PrintProviderError Function
static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Collect
    com.ms.wfc.data.Error          ErrItem = null;
    long                            nCount = 0;
    int                              i = 0;

    nCount = cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if ( nCount > 0)
    {
        // Collection ranges from 0 to nCount-1.
        for ( i=0;i<nCount; i++)
        {

```

```
        ErrItem = cnn1.getErrors().getItem(i);
        System.out.println("\t Error Number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription());
    }
}
// PrintIOError Function
static void PrintIOError(java.io.IOException je)
{
    System.out.println("Error: \n");
    System.out.println("\t Source: " + je.getClass() + "\n");
    System.out.println("\t Description: "+ je.getMessage() + "\n")
}
}
// EndFieldTypeJ
```

See Also

[Field Object](#) | [Type Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Type Property Example (Property) (VJ++)

This example demonstrates the [Type](#) property. It is a model of a utility for listing the names and types of a collection, like [Properties](#), [Fields](#), etc.

We do not need to open the [Recordset](#) to access its **Properties** collection; they come into existence when the **Recordset** object is instantiated. However, setting the [CursorLocation](#) property to **adUseClient** adds several [dynamic properties](#) to the **Recordset** object's **Properties** collection, making the example a little more interesting. For sake of illustration, we explicitly use the [Item](#) property to access each [Property](#) object.

```
// BegintypePropertyJ
import com.ms.wfc.data.*;
import java.io.* ;

public class TypePropertyX
{
    // The main entry point for the application.

    public static void main (String[] args)
    {
        TypePropertyX();
        System.exit(0);
    }

    // TypePropertyX function
    static void TypePropertyX()
    {
        // Define ADO Objects.
        Recordset rst = null;
        ADOProperty prop = null;

        // Declarations.
        BufferedReader in =
            new BufferedReader (new InputStreamReader(System.in));
        String strCnn = "DSN='Pubs';Provider='MSDASQL';Integrated Se
        String strMsg;
        int intIndex;
        int intDisplaysize = 15;
```

```

try
{
    rst = new Recordset();
    rst.setCursorLocation(AdoEnums.CursorLocation.CLIENT);
    for(intIndex = 0;
        intIndex <= rst.getProperties().getCount() - 1;intIn
    {
        prop = rst.getProperties().getItem(intIndex);
        switch(prop.getType())
        {
            case AdoEnums.DataType.BIGINT :
                strMsg = "adBigInt";
                break;
            case AdoEnums.DataType.BINARY :
                strMsg = "adBinary";
                break;
            case AdoEnums.DataType.BOOLEAN :
                strMsg = "adBoolean";
                break;
            case AdoEnums.DataType.BSTR :
                strMsg = "adBSTR";
                break;
            case AdoEnums.DataType.CHAPTER :
                strMsg = "adChapter";
                break;
            case AdoEnums.DataType.CHAR :
                strMsg = "adChar";
                break;
            case AdoEnums.DataType.CURRENCY :
                strMsg = "adCurrency";
                break;
            case AdoEnums.DataType.DATE :
                strMsg = "adDate";
                break;
            case AdoEnums.DataType.DBDATE :
                strMsg = "adDBDate";
                break;
            case AdoEnums.DataType.DBTIME :
                strMsg = "adDBTime";
                break;
            case AdoEnums.DataType.DBTIMESTAMP :
                strMsg = "adDBTimeStamp";
                break;
            case AdoEnums.DataType.DECIMAL :
                strMsg = "adDecimal";
                break;
            case AdoEnums.DataType.DOUBLE :
                strMsg = "adDouble";
                break;
            case AdoEnums.DataType.EMPTY :

```

```
        strMsg = "adEmpty";
        break;
case AdoEnums.DataType.ERROR :
    strMsg = "adError";
    break;
case AdoEnums.DataType.FILETIME :
    strMsg = "adFileTime";
    break;
case AdoEnums.DataType.GUID :
    strMsg = "adGUID";
    break;
case AdoEnums.DataType.IDISPATCH :
    strMsg = "adIDispatch";
    break;
case AdoEnums.DataType.INTEGER :
    strMsg = "adInteger";
    break;
case AdoEnums.DataType.IUNKNOWN :
    strMsg = "adIUnknown";
    break;
case AdoEnums.DataType.LONGVARBINARY :
    strMsg = "adLongVarBinary";
    break;
case AdoEnums.DataType.LONGVARCHAR :
    strMsg = "adLongVarChar";
    break;
case AdoEnums.DataType.LONGVARWCHAR :
    strMsg = "adLongVarWChar";
    break;
case AdoEnums.DataType.NUMERIC :
    strMsg = "adNumeric";
    break;
case AdoEnums.DataType.PROPVARIANT :
    strMsg = "adPropVariant";
    break;
case AdoEnums.DataType.SINGLE :
    strMsg = "adSingle";
    break;
case AdoEnums.DataType.SMALLINT :
    strMsg = "adSmallInt";
    break;
case AdoEnums.DataType.TINYINT :
    strMsg = "adTinyInt";
    break;
case AdoEnums.DataType.UNSIGNEDBIGINT :
    strMsg = "adUnsignedBigInt";
    break;
case AdoEnums.DataType.UNSIGNEDINT :
    strMsg = "adUnsignedInt";
```

```

        break;
    case AdoEnums.DataType.UNSIGNEDSMALLINT :
        strMsg = "adUnsignedSmallInt";
        break;
    case AdoEnums.DataType.UNSIGNEDTINYINT :
        strMsg = "adUnsignedTinyInt";
        break;
    case AdoEnums.DataType.USERDEFINED :
        strMsg = "adUserDefined";
        break;
    case AdoEnums.DataType.VARBINARY :
        strMsg = "adVarBinary";
        break;
    case AdoEnums.DataType.VARCHAR :
        strMsg = "adVarChar";
        break;
    case AdoEnums.DataType.VARIANT :
        strMsg = "adVariant";
        break;
    case AdoEnums.DataType.VARNUMERIC :
        strMsg = "adVarNumeric";
        break;
    case AdoEnums.DataType.VARWCHAR :
        strMsg = "adVarWChar";
        break;
    case AdoEnums.DataType.WCHAR :
        strMsg = "adWChar";
        break;
    default:
        strMsg = "*UNKNOWN*";
        break;
}
System.out.println("Property " +
    Integer.toString(intIndex) +
    " : " +
    prop.getName() +
    ", Type = " +
    strMsg);
if(intIndex % intDisplaysize == 0 && intIndex != 0)
{
    System.out.println("\nPress <Enter> to continue.");
    in.readLine();
}
}

System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{

```

```

        // Notify user of any errors that result from ADO.

        // As passing a Recordset, check for null pointer first.
        if (rst != null)
        {
            PrintProviderError(rst.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }

    // System read requires this catch.
    catch( java.io.IOException je)
    {
        PrintIOError(je);
    }

finally
{
    // Cleanup objects before exit.
    if (rst != null)
        if (rst.getState() == 1)
            rst.close();
}
}

// PrintProviderError Function
static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Colle
    com.ms.wfc.data.Error          ErrItem = null;
    long                          nCount = 0;
    int                            i = 0;

    nCount = cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if ( nCount > 0)
    {
        // Collection ranges from 0 to nCount-1.
        for ( i=0;i<nCount; i++)
        {
            ErrItem = cnn1.getErrors().getItem(i);
            System.out.println("\t Error Number: " + ErrItem.get
                + "\t" + ErrItem.getDescription());
        }
    }
}

```

```
    }  
  }  
  // PrintIOError Function  
  static void PrintIOError(java.io.IOException je)  
  {  
    System.out.println("Error: \n");  
    System.out.println("\t Source: " + je.getClass() + "\n");  
    System.out.println("\t Description: "+ je.getMessage() + "\n");  
  }  
}  
// EndTypePropertyJ
```

See Also

[Property Object](#) | [Type Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Update and CancelUpdate Methods Example (VJ++)

This example demonstrates the [Update](#) method in conjunction with the [CancelUpdate](#) method.

```
// BeginUpdateJ
import java.io.*;
import com.ms.wfc.data.*;

public class UpdateX
{
    // The main entry point of the application.

    public static void main (String[] args)
    {
        UpdateX();
        UpdateX2();
        System.exit(0);
    }
    // UpdateX Function

    static void UpdateX()
    {
        // Define ADO objects.
        Recordset rstEmployees = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            'Initial Catalog='Pubs';Integrated Security='SSPI'";
        String strOldFirst;
        String strOldLast;
        String strMessage;
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        try
        {
            // Open Recordset with names from Employees table.
            rstEmployees = new Recordset();
            rstEmployees.setCursorType(AdoEnums.CursorType.KEYSET);
            rstEmployees.setLockType(AdoEnums.LockType.OPTIMISTIC);
            rstEmployees.open("SELECT fname,lname FROM Employees " +
                "ORDER BY lname", strCnn, AdoEnums.CursorType.KEYSET,
```

```

        AdoEnums.LockType.OPTIMISTIC, AdoEnums.CommandType.TEXT)

// Store Original data.
strOldFirst = rstEmployees.getField("fname").getString();
strOldLast = rstEmployees.getField("lname").getString();

// Change data in edit buffer.
rstEmployees.getField("fname").setString("Linda");
rstEmployees.getField("lname").setString("Kobara");

// Show contents of buffer and get user input.
strMessage = "Edit in Progress :\n" +
    "\tOriginal Data = " + strOldFirst + " " + strOldLast +
    "\n\tData in Buffer = " +
    rstEmployees.getField("fname").getString() + " " +
    rstEmployees.getField("lname").getString() + "\n\n" +
    "Use Update to replace the original data with " +
    "the buffered data in the Recordset?Enter (Y/N)";

System.out.println(strMessage);
if(in.readLine().trim().equalsIgnoreCase("Y"))
    rstEmployees.update();
else
    rstEmployees.cancelUpdate();

// Show the resulting data.
System.out.println("Data in Recordset = " +
    rstEmployees.getField("fname").getString() +
    " " + rstEmployees.getField("lname").getString()+ "\n")

// Restore original data because this is a demonstration.
if(!(strOldFirst.equals(
    rstEmployees.getField("fname").getString()) &&
    strOldLast.equals(
    rstEmployees.getField("lname").getString()))
{
    rstEmployees.getField("fname").setString(strOldFirst);
    rstEmployees.getField("lname").setString(strOldLast);
    rstEmployees.update();
}
// Cleanup Objects before exit.
rstEmployees.close();

System.out.println("Press <Enter> to continue..");
in.readLine();
}
catch(ADOException ae)
{
    // Notify the user of any errors that result from ADO.

```

```

        // As passing a recordset, check for null pointer first.
        if(rstEmployees != null)
        {
            PrintProviderError(rstEmployees.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: "+ ae.getMessage());
        }
    }
    // System read requires this catch.
    catch(java.io.IOException je)
    {
        PrintIOError(je);
    }

    finally
    {
        // Cleanup objects before exit.
        if (rstEmployees != null)
            if (rstEmployees.getState() == 1)
                rstEmployees.close();
    }
}
// UpdateX2 Function

static void UpdateX2()
{
    // This example demonstrates the Update method in conjunction
    // with the AddNew method.

    // Define ADO Objects.
    Connection cnn1 = null;
    Recordset rstEmployees = null;

    // Declarations.
    String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
        "Initial Catalog='Pubs';Integrated Security='SSPI'";
    String strEmpID;
    String strOldFirst;
    String strOldLast;
    String strMessage;
    BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));

    try
    {
        // Open a connection.

```

```

cnn1 = new Connection();
cnn1.open(strCnn);

// Open Recordset with data from Employees table.
rstEmployees = new Recordset();
rstEmployees.setCursorType(AdoEnums.CursorType.KEYSET);
rstEmployees.setLockType(AdoEnums.LockType.OPTIMISTIC);
rstEmployees.open("employee", cnn1, AdoEnums.CursorType.KEY
    AdoEnums.LockType.OPTIMISTIC, AdoEnums.CommandType.TABLE

rstEmployees.addNew();
strEmpID="B-S55555M";
rstEmployees.getField("emp_id").setString(strEmpID);
rstEmployees.getField("fname").setString("Bill");
rstEmployees.getField("lname").setString("Sornsinn");

// Show contents of buffer and get user input.
strMessage = "AddNew in progress : " + "\n" +
    "\tData in Buffer = " +
    rstEmployees.getField("emp_id").getString() +
    " " + rstEmployees.getField("fname").getString() + " "
    rstEmployees.getField("lname").getString()+ "\n\n" +
    "Use Update to save buffer to recordset?Enter (Y/N)";
System.out.println(strMessage);
if(in.readLine().trim().equalsIgnoreCase("Y"))
{
    rstEmployees.update();
    // Go to the new record and show the resulting data.
    System.out.println("Data in recordset = " +
        rstEmployees.getField("emp_id").getString()+
        " " + rstEmployees.getField("fname").getString() +
        " " + rstEmployees.getField("lname").getString() + "
}
else
{
    rstEmployees.cancelUpdate();
    System.out.println("No new Record added.\n");
}
// Delete new data because this is a demonstration.
cnn1.execute(
    "DELETE FROM employee WHERE emp_id='" + strEmpID + "'");

// Cleanup Objects before exit
rstEmployees.close();

System.out.println("Press <Enter> to continue..");
in.readLine();
}
catch(ADOException ae)
{

```

```

        // Notify the user of any errors that result from ADO.

        // As passing a recordset, check for null pointer first.
        if(rstEmployees != null)
        {
            PrintProviderError(rstEmployees.getActiveConnection());
        }
        else
        {
            System.out.println("Exception: "+ ae.getMessage());
        }
    }
    // System read requires this catch.
    catch(java.io.IOException je)
    {
        PrintIOError(je);
    }

    finally
    {
        // Cleanup objects before exit.
        if (rstEmployees != null)
            if (rstEmployees.getState() == 1)
                rstEmployees.close();
    }
}
// PrintProviderError Function
static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Collect
    com.ms.wfc.data.Error          ErrItem = null;
    long                            nCount = 0;
    int                              i = 0;

    nCount = cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if ( nCount > 0)
    {
        // Collection ranges from 0 to nCount-1.
        for ( i=0;i<nCount; i++)
        {
            ErrItem = cnn1.getErrors().getItem(i);
            System.out.println("\t Error Number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription());
        }
    }
}
}

```

```
// PrintIOError Function
static void PrintIOError(java.io.IOException je)
{
    System.out.println("Error: \n");
    System.out.println("\t Source: " + je.getClass() + "\n");
    System.out.println("\t Description: "+ je.getMessage() + "\n")
}
}
// EndUpdateJ
```

See Also

[CancelUpdate Method](#) | [Update Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

UpdateBatch and CancelBatch Methods Example (VJ++)

This example demonstrates the [UpdateBatch](#) method in conjunction with the [CancelBatch](#) method.

```
// BeginUpdateBatchJ
import java.io.*;
import com.ms.wfc.data.*;

public class UpdateBatchX
{
    // The main entry point of the application.
    public static void main (String[] args)
    {
        UpdateBatchX();
        System.exit(0);
    }
    // UpdateBatchX Function

    static void UpdateBatchX()
    {
        // Define ADO Objects.
        Recordset rstTitles = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'";
        String strTitle;
        String strMessage;
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        try
        {
            rstTitles = new Recordset();
            rstTitles.setCursorType(AdoEnums.CursorType.KEYSET);
            rstTitles.setLockType(AdoEnums.LockType.BATCHOPTIMISTIC);
            rstTitles.open("Titles", strCnn, AdoEnums.CursorType.KEYSET
                AdoEnums.LockType.BATCHOPTIMISTIC,
                AdoEnums.CommandType.TABLE);

            rstTitles.moveFirst();
        }
    }
}
```

```

// Loop through recordset and ask user if she wants
// to change the type for the specified table.
while(!rstTitles.getEOF())
{
    if(rstTitles.getField("Type").getString().
        trim().equalsIgnoreCase("psychology"))
    {
        strTitle = rstTitles.getField("Title").getString();
        strMessage = "Title: " + strTitle + "\n" +
            "Change type to self_help?Enter (Y/N)";
        System.out.println(strMessage);
        if(in.readLine().trim().equalsIgnoreCase("Y"))
            rstTitles.getField("type").setString("self_help");
    }
    rstTitles.moveNext();
}
// Ask the user if she wants to commit to all the
// changes made above.
System.out.println("Save all changes?Enter (Y/N)");
if(in.readLine().trim().equalsIgnoreCase("Y"))
    rstTitles.updateBatch();
else
    rstTitles.cancelBatch();

// Print current data in recordset.
rstTitles.requery();
rstTitles.moveFirst();
while(!rstTitles.getEOF())
{
    System.out.println(rstTitles.getField("title").getString()
        " - " + rstTitles.getField("type").getString());
    rstTitles.moveNext();
}

// Restore original values because this is a demonstration.
rstTitles.moveFirst();
while(!rstTitles.getEOF())
{
    if(rstTitles.getField("type").getString().
        trim().equalsIgnoreCase("self_help"))
        rstTitles.getField("type").setString("psychology");
    rstTitles.moveNext();
}
rstTitles.updateBatch();

System.out.println("Press <Enter> to continue..");
in.readLine();
}
catch(AdoException ae)

```

```

{
    // Notify the user of any errors that result from ADO.

    // As passing a recordset, check for null pointer first.
    if(rstTitles != null)
    {
        PrintProviderError(rstTitles.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}

finally
{
    // Cleanup objects before exit.
    if (rstTitles != null)
        if (rstTitles.getState() == 1)
            rstTitles.close();
}
}

// PrintProviderError Function
static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Collect
    com.ms.wfc.data.Error          ErrItem = null;
    long                            nCount = 0;
    int                              i = 0;

    nCount = cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if ( nCount > 0)
    {
        // Collection ranges from 0 to nCount-1.
        for ( i=0;i<nCount; i++)
        {
            ErrItem = cnn1.getErrors().getItem(i);
            System.out.println("\t Error Number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription());
        }
    }
}

```

```
    }  
  }  
  // PrintIOError Function  
  static void PrintIOError(java.io.IOException je)  
  {  
    System.out.println("Error: \n");  
    System.out.println("\t Source: " + je.getClass() + "\n");  
    System.out.println("\t Description: "+ je.getMessage() + "\n")  
  }  
}  
// EndUpdateBatchJ
```

See Also

[CancelBatch Method](#) | [UpdateBatch Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Value Property Example (VJ++)

This example demonstrates the [Value](#) property with [Field](#) and [Property](#) objects by displaying field and property values for the *Employees* table.

```
// BeginValueJ
import java.io.*;
import com.ms.wfc.data.*;
import com.ms.com.*;

public class ValueX
{
    // Main Function
    public static void main (String[] args)
    {
        ValueX();
        System.exit(0);
    }
    static void ValueX()
    {
        // Define ADO Objects.
        Recordset rstEmployees = null;
        Field      fldLoop      = null;
        AdoProperty prpLoop     = null;

        // Declarations.
        String strCnn = "Provider='sqloledb';Data Source='MySqlServer'
            "Initial Catalog='Pubs';Integrated Security='SSPI'";
        int intLoop;
        BufferedReader in = new
            BufferedReader(new InputStreamReader(System.in));
        Variant varPropertyValue;
        String strMessage;

        try
        {
            // Open a Recordset with data from Employees table.
            rstEmployees = new Recordset();
            rstEmployees.open("employee", strCnn,
                AdoEnums.CursorType.FORWARDONLY, AdoEnums.LockType.READO
                AdoEnums.CommandType.TABLE);

            System.out.println("Field values in rstEmployees\n");

            // Enumerate the Fields collection of the Employees
```

```

// table.
for(intLoop = 0;
    intLoop<rstEmployees.getFields().getCount();intLoop++)
{
    fldLoop = rstEmployees.getFields().getItem(intLoop);
    // Because Value is the default property of a
    // Field object, the use of the actual keyword
    // here is optional.
    System.out.println("\t" + fldLoop.getName() + " = " +
        fldLoop.getValue());
}
System.out.println("\nPress <Enter> to continue..");
in.readLine();
System.out.println("Property values in rstEmployees\n");

// Enumerate the Properties collection of the
// Recordset object.
int intCount = 0;
for(intLoop = 0;
    intLoop<rstEmployees.getProperties().getCount();intLoop+
{
    prpLoop = rstEmployees.getProperties().getItem(intLoop);
    // Because Value is the default property of a
    // Field object, the use of the actual keyword
    // here is optional.
    strMessage = "\t" + prpLoop.getName() + " = ";
    varPropertyValue = prpLoop.getValue();
    short vttype =varPropertyValue.getvt();
    switch (vttype)
    {
    case Variant.VariantBoolean :
        {
            if (varPropertyValue.getBoolean())
                strMessage += "True";
            else
                strMessage += "False";
        }
        break;
    case Variant.VariantInt :
        strMessage += varPropertyValue.getInt();
        break;
    default :
        break;
    }
    System.out.println(strMessage);
    intCount++;
    // Loop used to display records
    if (intCount % 15 == 0)
    {
        System.out.println("\nPress <Enter> to continue..");
    }
}

```

```

        in.readLine();
        intCount = 0;
    }

}
// Cleanup objects before exit.
rstEmployees.close();
System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
// System read requires this catch.
catch(java.io.IOException je)
{
    PrintIOError(je);
}
catch(ADOException ae)
{
    // Notify the user of any errors that result from ADO.

    // As passing a recordset, check for null pointer first.
    if(rstEmployees != null)
    {
        PrintProviderError(rstEmployees.getActiveConnection());
    }
    else
    {
        System.out.println("Exception: " + ae.getMessage());
    }
}
}

finally
{
    // Cleanup objects before exit.
    if (rstEmployees != null)
        if (rstEmployees.getState() == 1)
            rstEmployees.close();
}
}

// PrintProviderError Function
static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Collect
    com.ms.wfc.data.Error          ErrItem = null;
    long                            nCount = 0;
    int                              i = 0;

    nCount = cnn1.getErrors().getCount();
}

```

```

// If there are any errors in the collection, print them.
if ( nCount > 0)
{
    // Collection ranges from 0 to nCount-1.
    for ( i=0;i<nCount; i++)
    {
        ErrItem = cnn1.getErrors().getItem(i);
        System.out.println("\t Error Number: " + ErrItem.getNumb
            + "\t" + ErrItem.getDescription());
    }
}
}
// PrintIOError Function
static void PrintIOError(java.io.IOException je)
{
    System.out.println("Error: \n");
    System.out.println("\t Source: " + je.getClass() + "\n");
    System.out.println("\t Description: "+ je.getMessage() + "\n")
}
}
// EndValueJ

```

See Also

[Field Object](#) | [Property Object](#) | [Value Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Version Property Example (VJ++)

This example uses the [Version](#) property of a [Connection](#) object to display the current ADO version. It also uses several [dynamic properties](#) to show:

- the current DBMS name and version.
- OLE DB version.
- [provider](#) name and version.
- [ODBC](#) version.
- ODBC driver name and version.

```
// BeginVersionJ
import com.ms.wfc.data.*;
import java.io.*;

public class VersionX
{
    // The main entry point of the application.
    public static void main (String[] args)
    {
        VersionX();
        System.exit(0);
    }
    // VersionX Function
    static void VersionX()
    {
        // Define ADO Objects.
        Connection cnn1 = null;

        // Declarations.
        String strCnn = "Driver={SQL Server};Server='MySqlServer';" +
            "User ID='MyUserID';Password='MyPassword';database
        String strVersionInfo;
        BufferedReader in = new
            BufferedReader(new InputStreamReader(System.in));
        try
        {
            // Open connection.
            cnn1 = new Connection();
            cnn1.open(strCnn);

            strVersionInfo = "\tADO Version:\t\t" +
                cnn1.getVersion().toString()+"\n"+
                "\tDBMS Name:\t\t" +
```

```

        cnn1.getProperties().getItem("DBMS Name").getString() +
        "\tDBMS Version:\t\t"+
        cnn1.getProperties().getItem("DBMS Version").getString()
        "\n" + "\tOLE DB Version:\t\t" +
        cnn1.getProperties().getItem("OLE DB Version").getString
        "\n" + "\tProvider Name:\t\t" +
        cnn1.getProperties().getItem("Provider Name").getString(
        "\n" + "\tProvider Version:\t" +
        cnn1.getProperties().getItem("Provider Version").
        getString() + "\n" + "\tDriver Name:\t\t" +
        cnn1.getProperties().getItem("Driver Name").getString()
        "\n" + "\tDriver Version:\t\t" +
        cnn1.getProperties().getItem("Driver Version").getString
        "\n" + "\tDriver ODBC Version:\t" +
        cnn1.getProperties().getItem(
        "Driver ODBC Version").getString()+ "\n";
        System.out.println("\n\n" + strVersionInfo);

        System.out.println("Press <Enter> to continue..");
        in.readLine();
    }
    // System read requires this catch.
    catch(java.io.IOException je)
    {
        PrintIOError(je);
    }
    catch(ADOException ae)
    {
        // Notify the user of any errors that result from ADO.

        // As passing a recordset, check for null pointer first.
        if(cnn1!= null)
        {
            PrintProviderError(cnn1);
        }
        else
        {
            System.out.println("Exception: " + ae.getMessage());
        }
    }
    finally
    {
        // Cleanup objects before exit.
        if (cnn1 != null)
            if (cnn1.getState() == 1)
                cnn1.close();
    }
}
// PrintProviderError Function

```

```

static void PrintProviderError(Connection cnn1)
{
    // Print Provider Errors from Connection Object.
    // ErrItem is an item object in the Connections Errors Collect
    com.ms.wfc.data.Error          ErrItem = null;
    long                          nCount = 0;
    int                            i = 0;

    nCount = cnn1.getErrors().getCount();

    // If there are any errors in the collection, print them.
    if ( nCount > 0)
    {
        // Collection ranges from 0 to nCount-1.
        for ( i=0;i<nCount; i++)
        {
            ErrItem = cnn1.getErrors().getItem(i);
            System.out.println("\t Error Number: " + ErrItem.getNumb
                + "\t" + ErrItem.getDescription());
        }
    }
}
// PrintIOError Function
static void PrintIOError(java.io.IOException je)
{
    System.out.println("Error: \n");
    System.out.println("\t Source: " + je.getClass() + "\n");
    System.out.println("\t Description: "+ je.getMessage() + "\n")
}
}
// EndVersionJ

```

See Also

[Connection Object](#) | [Version Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

ADO Code Examples in Microsoft JScript

Use the following code examples to learn how to use the ADO methods, properties, and events when writing in JScript.

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

Methods

- [AddNew Method Example](#)
- [Append and CreateParameter Methods Example](#)
- [Execute, Requery, and Clear Methods Example](#)
- [Find Method Example](#)
- [GetRows Method Example](#)

Properties

- [AbsolutePage, PageCount, and PageSize Properties Example](#)
- [AbsolutePosition and CursorLocation Properties Example](#)
- [ActiveCommand Property Example](#)
- [ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example](#)
- [ActualSize and DefinedSize Properties Example](#)
- [CacheSize Property Example](#)
- [Filter and RecordCount Properties Example](#)

See Also

[ADO Code Examples in Microsoft Visual Basic](#) | [ADO Code Examples in Microsoft Visual Basic Scripting Edition](#) | [ADO Code Examples in Microsoft Visual C++](#) | [ADO Code Examples in Microsoft Visual J++](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

AbsolutePage, PageCount, and PageSize Properties Example (JScript)

This example demonstrates the AbsolutePage, PageCount and PageSize properties. Cut and paste the following code to Notepad or another text editor, and save it as **AbsolutePageJS.asp**.

```
<!-- BeginAbsolutePageJS -->
<%@LANGUAGE="JScript" %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<html>

<head>
<title>AbsolutePage, PageSize, and PageSize Properties (JScript)</ti
<style>
<!--
BODY {
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
    }
.thead2 {
    background-color: #800000;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
    color: white;
    }
.tbody {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
    }
-->
</style>
</head>

<body bgcolor="White">
<h1>AbsolutePage, PageSize, and PageSize Properties (JScript)</h1>
```

```

<%
// connection and recordset variables
var Cnxn = Server.CreateObject("ADODB.Connection")
var strCnxn = "Provider='sqloledb';Data Source=" + Request.Serve
    "Initial Catalog='Northwind';Integrated Security='SSPI';
var rsEmployee = Server.CreateObject("ADODB.Recordset");
// display variables
var strMessage, iRecord, iPageCount;

try
{
    // open connection
    Cnxn.Open(strCnxn);

    // Open a recordset on the Employee table using
    // a client-side cursor to enable AbsolutePage property.
    rsEmployee.CursorLocation = adUseClient;
    rsEmployee.Open("employees", strCnxn, adOpenStatic, adLockOp

    // Set PageSize to five to display names and hire dates of f
    rsEmployee.PageSize = 5;
    iPageCount = rsEmployee.PageCount;

    // Write header information to the document
    Response.Write('<p align="center">There are ' + iPageCount);
    Response.Write(" pages, each containing ");
    Response.Write(rsEmployee.PageSize + " or fewer records.</p>");
    Response.Write('<table border="0" align="center">');
    Response.Write('<tr class="thead2">');
    Response.Write("<th>Page</th><th>Name</th><th>Hire Date</th>");

    for (var i=1; i<=iPageCount; i++)
    {
        rsEmployee.AbsolutePage = i;

        for (iRecord = 1; iRecord <= rsEmployee.PageSize; iRecor
        {
            strMessage = "";

            // Start a new table row.
            strMessage = '<tr class="tbody">';

            // First column in row contains page number on
            // first record of each page. Otherwise, the col
            // contains a non-breaking space.
            if (iRecord == 1)
                strMessage += "<td>Page " + i + " of " + rSE
            else
                strMessage += "<td>&nbsp;</td>";

```

```

        // First and last name are in first column.
        strMessage += "<TD>" + rsEmployee.Fields("FirstN
        strMessage += rsEmployee.Fields("LastName") + "

        // Hire date in second column.
        strMessage += "<td>" + rsEmployee.Fields("HireDa

        // End the row.
        strMessage += "</tr>";

        // Write line to document.
        Response.Write(strMessage);

        // Get next record.
        rsEmployee.MoveNext;

        if (rsEmployee.EOF)
            break;
    }
}

// Finish writing table.
Response.Write("</table>");
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // 'clean up
    if (rsEmployee.State == adStateOpen)
        rsEmployee.Close;
    if (Cnxn.State == adStateOpen)
        Cnxn.Close;
    rsEmployee = null;
    Cnxn = null;
}
%>

</body>

</html>
<!-- EndAbsolutePageJS -->

```

See Also

[AbsolutePage Property](#) | [PageCount Property](#) | [PageSize Property](#) | [Recordset](#)

Object

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

AbsolutePosition and CursorLocation Properties Example (JScript)

This example demonstrates how the [AbsolutePosition](#) property can track the progress of a loop that enumerates all the records of a [Recordset](#). It uses the [CursorLocation](#) property to enable the **AbsolutePosition** property by setting the [cursor](#) to a [client](#) cursor. Cut and paste the following code to Notepad or another text editor, and save it as **AbsolutePositionJS.asp**.

```
<!-- BeginAbsolutePositionJS -->
<%@LANGUAGE="JScript" %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<html>

<head>
<title>AbsolutePosition and CursorLocation Properties Example (JScript)
<style>
<!--
BODY {
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
    }
.thead2 {
    background-color: #800000;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
    color: white;
    }
.tbody {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
    }
-->
</style>
</head>

<body>
<h1>AbsolutePosition and CursorLocation Properties Example (JScript)
```

```

<%
// connection and recordset variables
var strCnxn = "Provider='sqloledb';Data Source=" + Request.Server
    "Initial Catalog='Northwind';Integrated Security='SSPI';
var rsEmployee = Server.CreateObject("ADODB.Recordset");
    // display string
var strMessage;

try
{
    // Open a recordset on the Employee table using
    // a client-side cursor to enable AbsolutePosition property.
    rsEmployee.CursorLocation = adUseClient;
    rsEmployee.Open("employees", strCnxn, adOpenStatic, adLockOp

    // Write beginning of table to the document.
    Response.Write('<table border="0" align="center">');
    Response.Write('<tr class="thead2">');
    Response.Write("<th>AbsolutePosition</th><th>Name</th><th>Hi

while (!rsEmployee.EOF)
{
    strMessage = "";

    // Start a new table row.
    strMessage = '<tr class="tbody">';

    // First column in row contains AbsolutePosition value.
    strMessage += "<td>" + rsEmployee.AbsolutePosition + " o

    // First and last name are in first column.
    strMessage += "<td>" + rsEmployee.Fields("FirstName") +
    strMessage += rsEmployee.Fields("LastName") + " " + "</t

    // Hire date in second column.
    strMessage += "<td>" + rsEmployee.Fields("HireDate") + "

    // End the row.
    strMessage += "</tr>";

    // Write line to document.
    Response.Write(strMessage);

    // Get next record.
    rsEmployee.MoveNext;
}

// Finish writing document.
Response.Write("</table>");
}

```

```
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // 'clean up
    if (rsEmployee.State == adStateOpen)
        rsEmployee.Close;
    rsEmployee = null;
}
%>

</html>
<!-- EndAbsolutePositionJS -->
```

See Also

[AbsolutePosition Property](#) | [CursorLocation Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActiveCommand Property Example (JScript)

This example demonstrates the [ActiveCommand](#) property. Cut and paste the following code to Notepad or another text editor, and save it as **ActiveCommandJS.asp**.

```
<!-- BeginActiveCommandJS -->
<%@LANGUAGE="JScript" %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<%
    // user input
    strName = new String(Request.Form("ContactName"))
%>

<html>

<head>
<title>ActiveCommand Property Example (JScript)</title>
<style>
<!--
BODY {
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
    }
-->
</style>
</head>

<body bgcolor="White">

<h1>ActiveCommand Property Example (JScript)</h1>

<%
if (strName.length > 0)
{
    // connection and recordset variables
    var Cnxn = Server.CreateObject("ADODB.Connection")
    var strCnxn = "Provider='sqloledb';Data Source=" + Request.S
        "Initial Catalog='Northwind';Integrated Security='SSPI';
```

```

var cmdContact = Server.CreateObject("ADODB.Command");
var rsContact = Server.CreateObject("ADODB.Recordset");
// display variables
var strMessage;

try
{
    // open connection
    Cnxn.Open(strCnxn);

    // Open a recordset using a command object
    cmdContact.CommandText = "SELECT ContactName FROM Custom";
    cmdContact.ActiveConnection = Cnxn;
    // create parameter and insert variable value
    cmdContact.Parameters.Append(cmdContact.CreateParameter(

rsContact = cmdContact.Execute();

while(!rsContact.EOF){
    // start new line
    strMessage = "<P>";

    // get data
    strMessage += rsContact("ContactName")

    // end the line
    strMessage += "</P>";

    // show data
    Response.Write(strMessage);

    // get next record
    rsContact.MoveNext;
}
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // 'clean up
    if (rsContact.State == adStateOpen)
        rsContact.Close;
    if (Cnxn.State == adStateOpen)
        Cnxn.Close;
    rsContact = null;
    Cnxn = null;
}
}

```

```
%>
```

```
<hr>
```

```
<form method="POST" action="ActiveCommandJS.asp">  
  <p align="left">Enter city of customer to find (e.g., Paris): <inp  
  <p align="left"><input type="submit" value="Submit" name="B1"><inp  
</form>  
</body>  
  
</html>  
<!-- EndActiveCommandJS -->
```

See Also

[ActiveCommand Property](#) | [Command Object](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example (JScript)

This example uses the [ActiveConnection](#), [CommandText](#), [CommandTimeout](#), [CommandType](#), [Size](#), and [Direction](#) properties to execute a stored procedure. Cut and paste the following code to Notepad or another text editor, and save it as **ActiveConnectionJS.asp**.

```
<!-- BeginActiveConnectionJS -->
<%@LANGUAGE="JScript"%>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<html>
<head>
    <title>ActiveConnection, CommandText, CommandTimeout, CommandTyp
<style>
<!--
BODY {
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
    }
.thead {
    background-color: #008080;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
    }
.thead2 {
    background-color: #800000;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
    }
.tbody {
    text-align: center;
    background-color: #f7efde;
```

```

    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
    }
-->
</style>
</head>

<body bgcolor="White">

<%
    var iRoyalty = parseInt(Request.Form("RoyaltyValue"));
    // check user input

    if (iRoyalty > -1)
    {
        // connection and recordset variables
        var Cnxn = Server.CreateObject("ADODB.Connection")
        var strCnxn = "Provider='sqloledb';Data Source=" + Request.S
            "Initial Catalog='pubs';Integrated Security='SSPI'";
        var cmdByRoyalty = Server.CreateObject("ADODB.Command");
        var rsByRoyalty = Server.CreateObject("ADODB.Recordset");
        var rsAuthor = Server.CreateObject("ADODB.Recordset");
        // display variables
        var filter, strMessage;

        try
        {
            // open connection
            Cnxn.Open(strCnxn);

            cmdByRoyalty.CommandText = "byroyalty";
            cmdByRoyalty.CommandType = adCmdStoredProc;
            cmdByRoyalty.CommandTimeout = 15;

            // The stored procedure called above is as follows:
            //     CREATE PROCEDURE byroyalty
            //     @percentage int
            //     AS
            //     SELECT au_id from titleauthor
            //     WHERE titleauthor.royaltyper = @percentage
            //     GO

            prmByRoyalty = Server.CreateObject("ADODB.Parameter");
            prmByRoyalty.Type = adInteger;
            prmByRoyalty.Size = 3;
            prmByRoyalty.Direction = adParamInput;
            prmByRoyalty.Value = iRoyalty;
            cmdByRoyalty.Parameters.Append(prmByRoyalty);

            cmdByRoyalty.ActiveConnection = Cnxn;

```

```

// recordset by Command - Execute
rsByRoyalty = cmdByRoyalty.Execute();

// recordset by Recordset - Open
rsAuthor.Open("Authors", Cnxn);

while (!rsByRoyalty.EOF)
{
    // set filter
    filter = "au_id='" + rsByRoyalty("au_id")
    rsAuthor.Filter = filter + "'";

    // start new line
    strMessage = "<P>";

    // get data
    strMessage += rsAuthor("au_fname") + " ";
    strMessage += rsAuthor("au_lname") + " ";

    // end line
    strMessage += "</P>";

    // show data
    Response.Write(strMessage);

    // get next record
    rsByRoyalty.MoveNext;
}
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // clean up
    if (rsByRoyalty.State == adStateOpen)
        rsByRoyalty.Close;
    if (rsAuthor.State == adStateOpen)
        rsAuthor.Close;
    if (Cnxn.State == adStateOpen)
        Cnxn.Close;
    rsByRoyalty = null;
    rsAuthor = null;
    Cnxn = null;
}
}
%>

```

```
<hr>
```

```
<form method="POST" action="ActiveConnectionJS.asp">  
  <p align="left">Enter royalty percentage to find (e.g., 40): <input  
  <p align="left"><input type="submit" value="Submit" name="B1"><input  
</form>  
&nbsp;
```

```
</body>
```

```
</html>
```

```
<!-- EndActiveConnectionJS -->
```

See Also

[ActiveCommand Property](#) | [Command Object](#) | [CommandText Property](#) | [CommandTimeout Property](#) | [CommandType Property](#) | [Connection Object](#) | [Direction Property](#) | [Parameter Object](#) | [Record Object](#) | [Recordset Object](#) | [Size Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

ActualSize and DefinedSize Properties Example (JScript)

This example uses the [ActualSize](#) and [DefinedSize](#) properties to display the defined size and actual size of a field. Cut and paste the following code to Notepad or another text editor, and save it as **ActualSizeJS.asp**.

```
<!-- BeginActualSizeJS -->
<%@LANGUAGE="JScript" %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E
<html>

<head>
  <title>ActualSize and DefinedSize Properties Example (JScript)</
<style>
<!--
body {
  font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
  BACKGROUND-COLOR:white;
  COLOR:black;
  }
.thead2 {
  background-color: #800000;
  font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
  font-size: x-small;
  color: white;
  }
.tbody {
  text-align: center;
  background-color: #f7efde;
  font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
  font-size: x-small;
  }
-->
</style>
</head>

<body bgcolor="White">

<h1>ADO ActualSize and DefinedSize Properties (JScript)</h1>
<%
  // connection and recordset variables
```

```

var Cnxn = Server.CreateObject("ADODB.Connection")
var strCnxn = "Provider='sqloledb';Data Source=" + Request.Server
              "Initial Catalog='Northwind';Integrated Security='SSPI';
var rsSuppliers = Server.CreateObject("ADODB.Recordset");
// display variables
var fld, strMessage;

try
{
    // open connection
    Cnxn.Open(strCnxn);

    // Open a recordset on the stores table
    rsSuppliers.Open("Suppliers", strCnxn);

    // build table headers
    Response.Write("<table>");
    Response.Write('<tr class="thead2"><th>Field Value</th>');
    Response.Write("<th>Defined Size</th>");
    Response.Write("<th>Actual Size</th></tr>");

    while (!rsSuppliers.EOF)
    {
        // start a new line
        strMessage = '<tr class="tbody">';

        // Display the contents of the chosen field with
        // its defined size and actual size
        fld = rsSuppliers("CompanyName");
        strMessage += '<td align="left">' + fld.Value + "</td>"
        strMessage += "<td>" + fld.DefinedSize + "</td>";
        strMessage += "<td>" + fld.ActualSize + "</td>";

        // end the line
        strMessage += "</tr>";

        // display data
        Response.Write(strMessage);

        // get next record
        rsSuppliers.MoveNext;
    }
    // close the table
    Response.Write("</table>");
}
catch (e)
{
    Response.Write(e.message);
}

```

```
finally
{
    // clean up
    if (rsSuppliers.State == adStateOpen)
        rsSuppliers.Close;
    if (Cnxn.State == adStateOpen)
        Cnxn.Close;
    rsSuppliers = null;
    Cnxn = null;
}
%>

</body>

</html>
<!-- EndActualSizeJS -->
```

See Also

[ActualSize Property](#) | [DefinedSize Property](#) | [Field Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

AddNew Method Example (JScript)

This example uses the [AddNew](#) method to create a new record with the specified name. Cut and paste the following code to Notepad or another text editor, and save it as **AddNewJS.asp**.

```
<!-- BeginAddNewJS -->
<%@LANGUAGE="JScript" %>
<!-- Include file for JScript ADO Constants -->
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<html>

<head>
    <title>Add New Method Example (JScript)</title>
<style>
<!--
body {
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
    }
-->
</style>
</head>

<body>
<h1>AddNew Method Example (JScript)</h1>

<%
    if (Request.Form("Addit") == "AddNew")
    {
        // connection and recordset variables
        var Cnxn = Server.CreateObject("ADODB.Connection")
        var strCnxn = "Provider='sqloledb';Data Source=" + Request.S
            "Initial Catalog='Northwind';Integrated Security='SSPI';
        var rsEmployee = Server.CreateObject("ADODB.Recordset");
        //record variables
        var FName = String(Request.Form("FirstName"));
        var LName = String(Request.Form("LastName"));

        try
        {
            // open connection
```

```

Cnxn.Open(strCnxn)

// open Employee recordset using client-side cursor
rsEmployee.CursorLocation = adUseClient;
rsEmployee.Open("Employees", strCnxn, adOpenKeyset, adLo

rsEmployee.AddNew();
rsEmployee("FirstName") = FName;
rsEmployee("LastName") = LName;
rsEmployee.Update;

// of course, you would normally do error handling here
Response.Write("New record added.")
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // clean up
    if (rsEmployee.State == adStateOpen)
        rsEmployee.Close;
    if (Cnxn.State == adStateOpen)
        Cnxn.Close;
    rsEmployee = null;
    Cnxn = null;
}
}
%>

```

```

<form method="post" action="AddNewJS.asp" id=form1 name=form1>
<table>
<tr>
    <td colspan="2">
        <h4>Please enter the record to add:</h4>
    </td>
</tr>
<tr>
    <td>
        First Name:
    </td>
    <td>
        <input name="FirstName" maxLength=20>
    </td>
</tr>
<tr>
    <td>
        Last Name:
    </td>

```

```
<td>
  <input name="LastName" size="30" maxLength=30>
</td>
</tr>
<tr>
  <td align="right">
    <input type="submit" value="Submit" name="Submit">
  </td>
  <td align="left">
    <input type="reset" value="Reset" name="Reset">
  </td>
</tr>
</table>
<input type="hidden" value="AddNew" name="Addit">
</form>
</body>
</HTML>
<!-- EndAddNewJS -->
```

See Also

[AddNew Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Append and CreateParameter Methods Example (JScript)

This example uses the [Append](#) and [CreateParameter](#) methods to execute a stored procedure with an input parameter. Cut and paste the following code to Notepad or another text editor, and save it as **AppendJS.asp**.

```
<!-- BeginAppendJS -->
<%@LANGUAGE="JScript" %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<html>
<head>
    <title>Append and CreateParameter Methods Example (JScript)</tit
<style>
<!--
body {
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
    }
-->
</style>
</head>

<body>
<h1>Append and CreateParameter Methods Example (JScript)</h1>
<%
    // verify user-input
    var iRoyalty = parseInt(Request.Form("RoyaltyValue"));
    if (iRoyalty > -1)
    {

        // connection, recordset and command variables
        var strCnxn = "Provider='sqloledb';Data Source=" + Request.S
            "Initial Catalog='pubs';Integrated Security='SSPI'";
        var Cnxn = Server.CreateObject("ADODB.Connection");
        var cmdByRoyalty = Server.CreateObject("ADODB.Command");
        var rsByRoyalty = Server.CreateObject("ADODB.Recordset");
        var rsAuthor = Server.CreateObject("ADODB.Recordset");
        // display variables
        var strMessage;
```

```

try
{
    // open connection and set cursor location
    Cnxn.Open(strCnxn);
    Cnxn.CursorLocation = adUseClient;

    // command object initial parameters
    cmdByRoyalty.CommandText = "byroyalty";
    cmdByRoyalty.CommandType = adCmdStoredProc;

    // create the new parameter and append to
    // the Command object's parameters collection
    var prmByRoyalty = cmdByRoyalty.CreateParameter("percent
    cmdByRoyalty.Parameters.Append(prmByRoyalty);
    prmByRoyalty.Value = iRoyalty;

    cmdByRoyalty.ActiveConnection = Cnxn;

    // execute command
    rsByRoyalty = cmdByRoyalty.Execute();

    // display results
    rsAuthor.Open("Authors", Cnxn);

    while (!rsByRoyalty.EOF)
    {
        rsAuthor.Filter = "au_id='" + rsByRoyalty.Fields("au

        // start new line
        strMessage = "<P>";

        // recordset data
        strMessage += rsAuthor.Fields("au_fname") + " ";
        strMessage += rsAuthor.Fields("au_lname") + " ";

        // end the line
        strMessage += "</P>";

        // show result
        Response.Write(strMessage);

        // et next record
        rsByRoyalty.MoveNext;
    }
}
catch (e)
{
    Response.Write(e.message);
}

```

```

    }
    finally
    {
        // clean up
        if (rsByRoyalty.State == adStateOpen)
            rsByRoyalty.Close;
        if (rsAuthor.State == adStateOpen)
            rsAuthor.Close;
        if (Cnxn.State == adStateOpen)
            Cnxn.Close;
        rsByRoyalty = null;
        rsAuthor = null;
        Cnxn = null;
    }
}
%>

<hr>

<form method="POST" action="AppendJS.asp" id=form1 name=form1>
  <p align="left">Enter royalty percentage to find (e.g., 40): <input
  <p align="left"><input type="submit" value="Submit" name="B1"><inp
</form>
&nbsp;

</body>

</html>
<!-- EndAppendJS -->

```

See Also

[Append Method](#) | [CreateParameter Method](#) | [Field Object](#) | [Fields Collection](#) | [Parameter Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

CacheSize Property Example (JScript)

This example uses the [CacheSize](#) property to show the difference in performance for an operation performed with and without a 30-record cache. Cut and paste the following code to Notepad or another text editor, and save it as **CacheSizeJS.asp**.

```
<!-- BeginCacheSizeJS -->
<%@ Language="JScript" %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<HTML>
<HEAD>
<title>CacheSize Property Example (JScript)</title>
<style>
<!--
body {
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
}
.thead2 {
    background-color: #800000;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
    color: white;
}
.tbody {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
}
-->
</style>
</HEAD>
<BODY>
<h1>CacheSize Property Example (JScript)</h1>
<%
    // connection and recordset variables
    var Cnxn = Server.CreateObject("ADODB.Connection")
```

```

var strCnxn = "Provider='sqloledb';Data Source=" + Request.Server
    "Initial Catalog='Northwind';Integrated Security='SSPI';
var rsCustomer = Server.CreateObject("ADODB.Recordset");
// caching variables
var Now = new Date();
var Start = Now.getTime();
var End, Cache, NoCache

try
{
    // open connection
    Cnxn.Open(strCnxn)

    // open a recordset on the Employee table using client-side
    rsCustomer.CursorLocation = adUseClient;
    rsCustomer.Open("Customers", strCnxn);

    // loop through the recordset 20 times
    for (var i=1; i<=20; i++)
    {
        rsCustomer.MoveFirst();
        while (!rsCustomer.EOF)
        {
            // do something with the record
            var strTemp = new String(rsCustomer("CompanyName"));
            rsCustomer.MoveNext();
        }
    }

    Now = new Date();
    End = Now.getTime();
    NoCache = End - Start;

    // cache records in groups of 30
    rsCustomer.MoveFirst();
    rsCustomer.CacheSize = 30;

    Now = new Date();
    Start = Now.getTime();

    // loop through the recordset 20 times
    for (var i=1; i<=20; i++)
    {
        rsCustomer.MoveFirst();
        while (!rsCustomer.EOF)
        {
            // do something with the record
            var strTemp = new String(rsCustomer("CompanyName"));
            rsCustomer.MoveNext();
        }
    }
}

```

```

    }

    Now = new Date();
    End = Now.getTime();
    var Cache = End - Start;
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // clean up
    if (rsCustomer.State == adStateOpen)
        rsCustomer.Close;
    if (Cnxn.State == adStateOpen)
        Cnxn.Close;
    rsCustomer = null;
    Cnxn = null;
}
%>

```

```

<table border="2">
  <tr class="thead2">
    <th>No Cache</th>
    <th>30 Record Cache</th>
  </tr>
  <tr class="tbody">
    <td><%=NoCache%></td>
    <td><%=Cache%></td>
  </tr>
</table>

```

```

</BODY>
</HTML>
<!-- EndCacheSizeJS -->

```

See Also

[CacheSize Property](#) | [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

Execute, Requery, and Clear Methods Example (JScript)

This example demonstrates the **Execute** method when run from both a [Command](#) object and a [Connection](#) object. It also uses the [Requery](#) method to retrieve current data in a [Recordset](#), and the [Clear](#) method to clear the contents of the [Errors](#) collection. (The **Errors** collection is accessed via the **Connection** object of the [ActiveConnection](#) property of the [Recordset](#).) Name the file **ExecuteJS.asp**.

```
<!-- BeginExecuteJS -->
<%@LANGUAGE="JScript"%>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<%
    strLastName = new String(Request.Form("AuthorLName"));

    if (strLastName.indexOf("undefined") > -1)
        strLastName = "";
%>

<html>

<head>
<title>Execute, Requery and Clear Methods Example (JScript)</title>
<style>
<!--
BODY {
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
    }
-->
</style>
</head>

<body bgcolor="White">
<h1>Execute, Requery and Clear Methods Example (JScript)</h1>
<%
    if (strLastName.length > 0)
    {
```

```

// command and recordset variables
var Connect = "Provider='sqloledb';Data Source=" + Request.S
    "Initial Catalog='pubs';Integrated Security='SSPI';";
var Cnxn = Server.CreateObject("ADODB.Connection");
var cmdAuthor = Server.CreateObject("ADODB.Command");
var rsAuthor = Server.CreateObject("ADODB.Recordset");
var rsAuthor2 = Server.CreateObject("ADODB.Recordset");
var SQLAuthor2, strMessage, strMessage2;
var Err, ErrCount;

try
{
    // open connection
    Cnxn.Open(Connect);

    // command object parameters
    cmdAuthor.CommandText = "SELECT * FROM Authors WHERE au_
cmdAuthor.Parameters.Append(cmdAuthor.CreateParameter("L
cmdAuthor.ActiveConnection = Cnxn;

    // recordset from command.execute
    rsAuthor = cmdAuthor.Execute();

    // recordset from connection.execute
    SQLAuthor2 = "SELECT * FROM Authors";
    rsAuthor2 = Cnxn.Execute(SQLAuthor2);

    // check for errors
    ErrCount = Cnxn.errors.count;
    if(ErrCount !== 0) //write the errors
    {
        for(Err = 0; Err = ErrCount; Err++){
            Err = Cnxn.errors.item;
            Response.Write(Err);
        }
        // clean out any existing errors
        Cnxn.Errors.Clear;
    }

    // show the data
    Response.Write("<HR><HR>");

    // first recordset
    Response.Write("<b>Command.Execute results</b>")
    while (!rsAuthor.EOF)
    {
        // build output string by starting a new line
        strMessage = "<P>";
        strMessage += "<br>";
    }
}

```

```

        // recordset data
        strMessage += rsAuthor("au_fname") + " ";
        strMessage += rsAuthor("au_lname") + " ";

        // end the line
        strMessage += "</P>";

        // show the results
        Response.Write(strMessage);

        // get next record
        rsAuthor.MoveNext;
    }

    Response.Write("<HR><HR>");

    // second recordset
    Response.Write("<b>Connection.Execute results</b>")
    while (!rsAuthor2.EOF)
    {
        // start a new line
        strMessage2 = "<P>";

        // first and last name are in first column
        strMessage2 += rsAuthor2("au_fname") + " ";
        strMessage2 += rsAuthor2("au_lname") + " ";

        // end the line
        strMessage2 += "</P>";

        // show results
        Response.Write(strMessage2);

        // get next record
        rsAuthor2.MoveNext;
    }
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // clean up
    if (rsAuthor.State == adStateOpen)
        rsAuthor.Close;
    if (rsAuthor2.State == adStateOpen)
        rsAuthor2.Close;
    if (Cnxn.State == adStateOpen)

```

```
        Cnxn.Close;
        rsAuthor1 = null;
        rsAuthor2 = null;
        Cnxn = null;
    }
}
%>

<hr>

<form method="POST" action="ExecuteJS.asp" id=form1 name=form1>
  <p align="left">Enter last name of author to find (e.g., Ringer):
  <p align="left"><input type="submit" value="Submit" name="B1"><inp
</form>
</body>

</html>
<!-- EndExecuteJS -->
```

See Also

[Clear Method](#) | [Command Object](#) | [Connection Object](#) | [Error Object](#) | [Execute Method \(ADO Command\)](#) | [Execute Method \(ADO Connection\)](#) | [Recordset Object](#) | [Requery Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Filter and RecordCount Properties Example (JScript)

This example opens a **Recordset** on the Companies table of the Northwind database and then uses the [Filter](#) property to limit the records visible to those where the CompanyName field starts with the letter D. Cut and paste the following code to Notepad or another text editor, and save it as **FilterJS.asp**.

```
<!-- BeginFilterJS -->
<%@ Language=JavaScript %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<html>

<head>
<title>ADO Recordset.Filter Example</title>
<style>
<!--
BODY {
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
}
.thead {
    background-color: #008080;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
    color: white;
}
.thead2 {
    background-color: #800000;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
    color: white;
}
.tbody {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
    font-size: x-small;
}
-->
```

```

</style>
</head>

<body bgcolor="White">

<h1>ADO Recordset.Filter Example</h1>
<!-- Page text goes here -->
<%
    // connection and recordset variables
    var Cnxn = Server.CreateObject("ADODB.Connection")
    var strCnxn = "Provider='sqloledb';Data Source=" + Request.Server
        "Initial Catalog='Northwind';Integrated Security='SSPI';
    var rsCustomers = Server.CreateObject("ADODB.Recordset");
    var SQLCustomers = "select * from Customers;";
    // record variables
    var fld, filter
    var showBlank = " ";
    var showNull = "-NULL-";

    try
    {
        //open connection
        Cnxn.Open(strCnxn);

        // create recordset client-side using object refs
        rsCustomers.ActiveConnection = Cnxn;
        rsCustomers.CursorLocation = adUseClient;
        rsCustomers.CursorType = adOpenKeyset;
        rsCustomers.LockType = adLockOptimistic;
        rsCustomers.Source = SQLCustomers;
        rsCustomers.Open();

        rsCustomers.MoveFirst();

        //set filter
        filter = "CompanyName LIKE 'b*';
        rsCustomers.Filter = filter

        if (rsCustomers.RecordCount == 0) {
            Response.Write("No records matched ");
            Response.Write (SQLCustomers + "So cannot make table..."
            Cnxn.Close();
            Response.End
        }
        else {
            // show the data
            Response.Write('<table width="100%" border="2">');
            while(!rsCustomers.EOF) {
                Response.Write('<tr class="tbody">');
                for (var thisField = 0; thisField < rsCustomers.Fiel

```

```

        fld = rsCustomers(thisField);
        fldValue = fld.Value;
        if (fldValue == null)
            fldValue = showNull;
        if (fldValue == "")
            thisField=showBlank;
        Response.Write("<td>" + fldValue + "</td>")
    }
    rsCustomers.MoveNext();
    Response.Write("</tr>");
}
// close the table
Response.Write("</table>");
}
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // clean up
    if (rsCustomers.State == adStateOpen)
        rsCustomers.Close;
    if (Cnxn.State == adStateOpen)
        Cnxn.Close;
    rsCustomers = null;
    Cnxn = null;
}
%>

</body>

</html>
<!-- EndFilterJS -->

```

See Also

[Filter Property](#) | [RecordCount Property](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 Samples 

Find Method Example (JScript)

This example uses the [Recordset](#) object's [Find](#) method to locate and display the companies in the *Northwind* database whose name begins with the letter G. Cut and paste the following code to Notepad or another text editor, and save it as **FindJS.asp**.

```
<!-- BeginFindJS -->
<%@ Language=JavaScript %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<html>

<head>
<title>ADO Recordset.Find Example</title>
<style>
<!--
BODY {
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
}
.thead {
    background-color: #008080;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
}
.thead2 {
    background-color: #800000;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
}
.tbody {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
}
-->
</style>
</head>
```

```

<body bgcolor="white">

<h1>ADO Recordset.Find Example</h1>
<%
    // connection and recordset variables
    var Cnxn = Server.CreateObject("ADODB.Connection");
    var strCnxn = "Provider='sqloledb';Data Source=" + Request.Serve
        "Initial Catalog='Northwind';Integrated Security='SSPI'";
    var rsCustomers = Server.CreateObject("ADODB.Recordset");
        // display string
    var strMessage;
    var strFind;

    try
    {
        // open connection
        Cnxn.Open(strCnxn);

        //create recordset using object refs
        SQLCustomers = "select * from Customers;";

        rsCustomers.ActiveConnection = Cnxn;
        rsCustomers.CursorLocation = adUseClient;
        rsCustomers.CursorType = adOpenKeyset;
        rsCustomers.LockType = adLockOptimistic;
        rsCustomers.Source = SQLCustomers;

        rsCustomers.Open();
        rsCustomers.MoveFirst();

        //find criteria
        strFind = "CompanyName like 'g%'"
        rsCustomers.Find(strFind);

        if (rsCustomers.EOF) {
            Response.Write("No records matched ");
            Response.Write(SQLCustomers & "So cannot make table...")
            Cnxn.Close();
            Response.End();
        }
        else {
            Response.Write('<table width="100%" border="2">');
            Response.Write('<tr class="thead2">');
            // Put Headings On The Table for each Field Name
            for (thisField = 0; thisField < rsCustomers.Fields.Count
                fieldObject = rsCustomers.Fields(thisField);
                Response.Write('<th width="' + Math.floor(100 / rsCu
            }
            Response.Write("</tr>");

```

```

        while (!rsCustomers.EOF) {
            Response.Write('<tr class="tbody">');
            for(thisField=0; thisField<rsCustomers.Fields.Count;
                fieldObject = rsCustomers.Fields(thisField);
                strField = fieldObject.Value;
                if (strField == null)
                    strField = "-Null-";
                if (strField == "")
                    strField = "";
                Response.Write("<td>" + strField + "</td>");
            }
            rsCustomers.Find(strFind, 1, adSearchForward)
            Response.Write("</tr>");
        }
        Response.Write("</table>");
    }
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // clean up
    if (rsCustomers.State == adStateOpen)
        rsCustomers.Close;
    if (Cnxn.State == adStateOpen)
        Cnxn.Close;
    rsCustomers = null;
    Cnxn = null;
}
%>

</body>

</html>
<!-- EndFindJS -->

```

See Also

[Find Method](#) | [Recordset Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 Samples 

GetRows Method Example (JScript)

This example uses the [GetRows](#) method to retrieve all rows of the *Customers* table from a [Recordset](#) and to fill an array with the resulting data. The **GetRows** method will return fewer than the desired number of rows in two cases: either if [EOF](#) has been reached, or if **GetRows** tried to retrieve a record that was deleted by another user. The function returns **False** only if the second case occurs. Cut and paste the following code to Notepad or another text editor, and save it as **GetRowsJS.asp**.

```
<!-- BeginGetRowsJS -->
<%@ LANGUAGE="JScript" %>
<%// use this meta tag instead of adojavas.inc%>
<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2E

<html>

<head>
<title>ADO Recordset.GetRows Example (JScript)</title>
<style>
<!--
BODY {
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    BACKGROUND-COLOR:white;
    COLOR:black;
    }
.thead {
    background-color: #008080;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
    }
.thead2 {
    background-color: #800000;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    color: white;
    }
.tbody {
    text-align: center;
    background-color: #f7efde;
    font-family: 'Verdana','Arial','Helvetica',sans-serif;
    font-size: x-small;
    }
```

```

-->
</style>
</head>

<body bgcolor="white">

<h1>ADO Recordset.GetRows Example (JScript)</h1>
  <!-- Page text goes here -->
<%
    var Connect = "Provider='sqloledb';Data Source=" + Request.S
        "Initial Catalog='Northwind';Integrated Security='SSPI';
var mySQL = "select * from customers;";
var showblank = " ";
var shownull = "-null-";

var connTemp = Server.CreateObject("ADODB.Connection");

try
{
    connTemp.Open(Connect);
    var rsTemp = Server.CreateObject("ADODB.Recordset");
    rsTemp.ActiveConnection = connTemp;
    rsTemp.CursorLocation = adUseClient;
    rsTemp.CursorType = adOpenKeyset;
    rsTemp.LockType = adLockOptimistic;
    rsTemp.Open(mySQL);

    rsTemp.MoveFirst();

    if (rsTemp.RecordCount == 0)
    {
        Response.Write("No records matched ");
        Response.Write (mySQL & "So cannot make table...");
        connTemp.Close();
        Response.End();
    } else
    {
        Response.Write('<table width="100%" border="2">');
        Response.Write('<tr class="thead2">');

        // Headings On The Table for each Field Name
        for (var i=0; i<rsTemp.Fields.Count; i++)
        {
            fieldObject = rsTemp.fields(i);
            Response.Write('<td width="' + Math.floor(100 / rsTe
        }

        Response.Write("</tr>");

        // JScript doesn't support multi-dimensional arrays

```

```

        // so we'll convert the returned array to a single
        // dimensional JScript array and then display the data.
        tempArray = rsTemp.GetRows();
        recArray = tempArray.toArray();

        var col = 1;
        var maxCols = rsTemp.Fields.Count;

        for (var thisField=0; thisField<recArray.length; thisFie
        {
            if (col == 1)
                Response.Write('<tr class="tbody">');
            if (recArray[thisField] == null)
                recArray[thisField] = shownull;
            if (recArray[thisField] == "")
                recArray[thisField] = showblank;
            Response.Write("<td>" + recArray[thisField] + "</td>
            col++
            if (col > maxCols)
            {
                Response.Write("</tr>");
                col = 1;
            }
        }
        Response.Write("</table>");
    }
}
catch (e)
{
    Response.Write(e.message);
}
finally
{
    // clean up
    if (rsTemp.State == adStateOpen)
        rsTemp.Close;
    if (connTemp.State == adStateOpen)
        connTemp.Close;
    rsTemp = null;
    connTemp = null;
}
%>

</body>

</html>
<!-- EndGetRowsJS -->

```

See Also

[GetRows Method](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

RDS API Reference

This section of the ADO documentation contains topics for each RDS object, property, method, and event. For more information, search for a specific topic in the index or refer to the following topics:

- [RDS Objects](#)
- [RDS Properties](#)
- [RDS Methods](#)
- [RDS Events](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

RDS Objects

[DataControl \(RDS\)](#)

Binds a data query **Recordset** to one or more controls (for example, a text box, grid control, or combo box) to display the **Recordset** data on a Web page.

[DataFactory \(RDSServer\)](#)

Implements methods that provide read/write data access to specified data sources for client-side applications.

[DataSpace \(RDS\)](#)

Creates client-side proxies to custom business objects located on the middle tier.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

DataControl Object (RDS)

Binds a data query [Recordset](#) to one or more controls (for example, a text box, grid control, or combo box) to display the **Recordset** data on a Web page.



Syntax

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID="Dat
  <PARAM NAME="Connect" VALUE="DSN=DSNName;UID=usr;PWD=pw;">
  <PARAM NAME="Server" VALUE="http://awebsrvr">
  <PARAM NAME="SQL" VALUE="QueryText">
</OBJECT>
```

Remarks

The [class ID](#) for the **RDS.DataControl** object is BD96C556-65A3-11D0-983A-00C04FC29E33.

Note If you get an error that an [RDS.DataSpace](#) or **RDS.DataControl** object doesn't load, make sure you are using the correct class ID. The class IDs for these objects have changed from version 1.0 and 1.1.

For a basic scenario, you need to set only the **SQL**, **Connect**, and **Server** properties of the **RDS.DataControl** object, which will automatically call the default [business object](#), [RDS.Server.DataFactory](#).

All the properties in the **RDS.DataControl** are optional because custom business objects can replace their functionality.

Note If you query for multiple results, only the first [Recordset](#) is returned. If multiple result sets are needed, assign each to its own **DataControl**. An example of a query for multiple results could be the following:
"Select * from Authors, Select * from Topics"

Adding "DFMode=20;" to your connection string when using the **RDS.DataControl** object can improve your server's performance when updating

data. With this setting, the **RDS.Server.DataFactory** object on the server uses a less resource-intensive mode. However, the following features are not available in this configuration:

- Using parameterized queries.
- Getting parameter or column information before calling the **Execute** method.
- Setting **Transact Updates** to **True**.
- Getting row status.
- Calling the [Resync](#) method.
- Refreshing (explicitly or automatically) via the [Update Resync](#) property.
- Setting **Command** or [Recordset](#) properties.
- Using **adCmdTableDirect**.

The **RDS.DataControl** object runs in [asynchronous](#) mode by default. If you require synchronous execution for your application, set the [ExecuteOptions](#) parameter equal to **adcExecSync** and the [FetchOptions](#) parameter equal to **adcFetchUpFront**, as shown in the following example.

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID="DataControl"
  <PARAM NAME="Connect" VALUE="DSN=DSNName;UID=usr;PWD=pw;">
  <PARAM NAME="Server" VALUE="http://awebsrvr">
  <PARAM NAME="SQL" VALUE="QueryText">
  <PARAM NAME="ExecuteOptions" VALUE="1">
  <PARAM NAME="FetchOptions" VALUE="1">
</OBJECT>
```

Use one **RDS.DataControl** object to link the results of a single query to one or more visual controls. For example, suppose you code a query requesting customer data such as Name, Residence, Place of Birth, Age, and Priority Customer Status. You can use a single **RDS.DataControl** object to display a customer's Name, Age, and Region in three separate text boxes; Priority Customer Status in a check box; and all the data in a grid control.

Use different **RDS.DataControl** objects to link the results of multiple queries to different visual controls. For example, suppose you use one query to obtain information about a customer, and a second query to obtain information about merchandise that the customer has purchased. You want to display the results of the first query in three text boxes and one check box, and the results of the second query in a grid control. If you use the default business object

(**RDS.Server.DataFactory**), you need to do the following:

- Add two **RDS.DataControl** objects to your Web page.
- Write two queries, one for each **SQL** property of the two **RDS.DataControl** objects. One **RDS.DataControl** object will contain an SQL query requesting customer information; the second will contain a query requesting a list of merchandise the customer has purchased.
- In each of the bound controls' **OBJECT** tags, specify the **DATAFLD** value to set the values for the data you want to display in each visual control.

There is no count restriction on the number of **RDS.DataControl** objects that you can embed via **OBJECT** tags on a single Web page.

When you define the **RDS.DataControl** object on a Web page, use nonzero **Height** and **Width** values such as 1 (to avoid the inclusion of extra space).

Remote Data Service client components are already included as part of Internet Explorer 4.0; therefore, you don't need to include a **CODEBASE** parameter in your **RDS.DataControl** object tag.

With Internet Explorer 4.0 or later, you can bind to data by using HTML controls and ActiveX® controls only if they are marked as apartment model controls.

Microsoft Visual Basic Users The **RDS.DataControl** is used only in Web-based applications. A Visual Basic client application has no need for it.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

DataControl Object (RDS)

Properties, Methods, and Events

Properties

[Connect Property \(RDS\)](#)

[ExecuteOptions Property \(RDS\)](#)

[FetchOptions Property \(RDS\)](#)

[FilterColumn Property \(RDS\)](#)

[FilterCriterion Property \(RDS\)](#)

[FilterValue Property \(RDS\)](#)

[Handler Property \(RDS\)](#)

[InternetTimeout Property \(RDS\)](#)

[ReadyState Property \(RDS\)](#)

[Recordset, SourceRecordset Properties \(RDS\)](#)

[Server Property \(RDS\)](#)

[SortColumn Property \(RDS\)](#)

[SortDirection Property \(RDS\)](#)

[SQL Property \(RDS\)](#)

[URL Property \(RDS\)](#)

Methods

[Cancel Method \(RDS\)](#)

[CancelUpdate Method \(RDS\)](#)

[CreateRecordset Method \(RDS\)](#)

[MoveFirst, MoveLast, MoveNext, and MovePrevious Methods \(RDS\)](#)

[Refresh Method \(RDS\)](#)

[Reset Method \(RDS\)](#)

[SubmitChanges Method \(RDS\)](#)

Events

[OnError Event](#)

[OnReadyStateChange Event](#)

See Also

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

DataFactory Object (RDSServer)

This default server-side [business object](#) implements methods that provide read/write data access to specified data sources for [client-side](#) applications.



Remarks

The **RDSServer.DataFactory** object is designed as a server-side Automation object that receives client requests. In an Internet implementation, it resides on a [Web server](#) and is instantiated by the [ADISAPI component](#). The **RDSServer.DataFactory** object provides read and write access to specified data sources, but doesn't contain any validation or [business rules](#) logic.

If you use a method that is available in both the **RDSServer.DataFactory** and [RDS.DataControl](#) objects, Remote Data Service uses the **RDS.DataControl** version by default. The default assumes a basic programming scenario, where the **RDSServer.DataFactory** serves as a generic server-side business object.

If you want your Web application to handle task-specific server-side processing, you can replace the **RDSServer.DataFactory** with a custom business object.

You can create server-side business objects that call the **RDSServer.DataFactory** methods, such as [Query](#) and [CreateRecordset](#). This is helpful if you want to add functionality to your business objects, but take advantage of existing Remote Data Service technologies.

The [class ID](#) for the **RDSServer.DataFactory** object is 9381D8F5-0288-11D0-9501-00AA00B911A5.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

DataFactory Object (RDSServer) Properties, Methods, and Events

Properties

None.

Methods

[ConvertToString Method \(RDS\)](#)

Converts a recordset into a MIME64 string.

[CreateRecordset Method \(RDS\)](#)

Creates and returns an empty recordset.

[Query Method \(RDS\)](#)

Execute the request and create an advanced data rowset.

[SubmitChanges Method \(RDS\)](#)

Given a recordset with pending changes, this method submits them to the database identified in the connection string.

Events

None.

See Also

Applies To: [DataFactory Object \(RDSServer\)](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

DataSpace Object (RDS)

Creates [client-side proxies](#) to custom [business objects](#) located on the [middle tier](#).



Remarks

Remote Data Service needs business object proxies so that client-side [components](#) can communicate with business objects located on the middle tier. Proxies facilitate the packaging, unpacking, and transport ([marshaling](#)) of the application's [Recordset](#) data across process or machine boundaries.

Remote Data Service uses the **RDS.DataSpace** object's [CreateObject](#) method to create business object proxies. The business object proxy is dynamically created whenever an instance of its middle-tier business object counterpart is created. Remote Data Service supports the following protocols: HTTP, HTTPS (HTTP Secure Sockets), DCOM, and in-process (client components and the business object reside on the same computer).

Note RDS behaves in a "stateless" manner when the **RDS.DataSpace** object uses the HTTP or HTTPS protocols. That is, any internal information about a client request is discarded after the server returns a response.

Although the business object appears to exist for the lifetime of the business object proxy, the business object actually exists only until a response is sent to a request. When a request is issued (that is, a method is invoked on the business object), the proxy opens a new connection to the server and the server creates a new instance of the business object. After the business object responds to the request, the server destroys the business object and closes the connection.

This behavior means you cannot pass data from one request to another using a business object property or variable. You must employ some other mechanism, such as a file or a method argument, to persist state data.

The [class ID](#) for the **RDS.DataSpace** object is BD96C556-65A3-11D0-983A-

00C04FC29E36.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

DataSpace Object (RDS) Properties, Methods, and Events

Properties

[InternetTimeout Property \(RDS\)](#)

Methods

[CreateObject Method \(RDS\)](#)

Events

None.

See Also

Applies To: [DataSpace Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

RDS Properties

[Connect \(RDS\)](#)

Indicates the database name from which the query and update operations are run.

[ExecuteOptions \(RDS\)](#)

Indicates whether asynchronous execution is enabled.

[FetchOptions \(RDS\)](#)

Indicates the type of asynchronous fetching.

[FilterColumn \(RDS\)](#)

Indicates the column on which to evaluate the filter criteria.

[FilterCriterion \(RDS\)](#)

Indicates the evaluation operator to use in the filter value.

[FilterValue \(RDS\)](#)

Indicates the value to filter records.

[Handler \(RDS\)](#)

Indicates the name of a server-side customization program (*handler*) that extends the functionality of the **RDS***Server.DataFactory*, and any parameters used by the *handler*.

[InternetTimeout \(RDS\)](#)

Indicates the number of milliseconds to wait before a request times out.

[ReadyState \(RDS\)](#)

Indicates the progress of a **DataControl** object as it fetches data into its **Recordset** object.

[Recordset and SourceRecordset \(RDS\)](#)

Indicates the **Recordset** object returned from a custom business object.

[Server \(RDS\)](#)

Indicates the Internet Information Services (IIS) name and communication protocol.

[SortColumn \(RDS\)](#)

Indicates by which column to sort the records.

[SortDirection \(RDS\)](#)

Indicates whether a sort order is ascending or descending.

[SQL \(RDS\)](#)

Indicates the query string used to retrieve the **Recordset**.

[URL \(RDS\)](#)

Indicates a string that contains a relative or absolute URL.

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

Connect Property (RDS)

Indicates the database name from which the query and update operations are run.

You can set the **Connect** property at design time in the [RDS.DataControl](#) object's OBJECT tags, or at run time in scripting code (for instance, VBScript).

Syntax

Design time: `<PARAM NAME="Connect" VALUE="ConnectionString">`

Run time: `DataControl.Connect = "ConnectionString"`

Parameters

ConnectionString

A valid connection string. For more general information about connection strings, see the [ConnectionString](#) property or your provider documentation.

Note Specifying MS Remote as the provider for the **RDS.DataControl** would create a four-tier scenario. Scenarios greater than three tiers have not been tested and should not be needed.

DataControl

An [object variable](#) that represents an **RDS.DataControl** object.

See Also

[VBScript Example](#)

[Query Method \(RDS\)](#) | [Refresh Method \(RDS\)](#) | [SubmitChanges Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

ExecuteOptions Property (RDS)

Indicates whether [asynchronous](#) execution is enabled.

Settings and Return Values

Sets or returns one of the following values.

Constant	Description
adcExecSync	Executes the next refresh of the Recordset synchronously .
adcExecAsync	Default. Executes the next refresh of the Recordset asynchronously.

Note Each [client-side](#) executable file that uses these constants must provide declarations for them. You can cut and paste the constant declarations that you want from the file `Adcvbs.inc`, located in the `C:\Program Files\Common Files\System\MSADC` folder.

Remarks

If **ExecuteOptions** is set to **adcExecAsync**, then this asynchronously executes the next **Refresh** call on the [RDS.DataControl](#) object's **Recordset**.

If you try to call [Reset](#), [Refresh](#), [SubmitChanges](#), [CancelUpdate](#), or [Recordset](#) while another asynchronous operation that might change the [RDS.DataControl](#) object's **Recordset** is executing, an error occurs.

If an error occurs during an asynchronous operation, the **RDS.DataControl** object's [ReadyState](#) value changes from **adcReadyStateLoaded** to **adcReadyStateComplete**, and the **Recordset** property value remains *Nothing*.

See Also

[VBScript Example](#)

[Cancel Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

FetchOptions Property (RDS)

Indicates the type of [asynchronous](#) fetching.

Setting and Return Values

Sets or returns one of the following values.

Constant	Description
adcFetchUpFront	All the records of the Recordset are fetched before control is returned to the application. The complete Recordset is fetched before the application is allowed to do anything with it.
adcFetchBackground	Control can return to the application as soon as the first batch of records has been fetched. A subsequent read of the Recordset that attempts to access a record not fetched in the first batch will be delayed until the sought record is actually fetched, at which time control returns to the application.
adcFetchAsync	Default. Control returns immediately to the application while records are fetched in the background. If the application attempts to read a record that hasn't yet been fetched, the record closest to the sought record will be read and control will return immediately, indicating that the current end of the Recordset has been reached. For example, a call to MoveLast will move the current record position to the last record actually fetched, even though more records will continue to populate the Recordset .

Note Each [client-side](#) executable file that uses these constants must provide declarations for them. You can cut and paste the constant declarations you want from the file Adcvbs.inc, located in the C:\Program Files\Common Files\System\MSADC folder.

Remarks

In a Web application, you will usually want to use **adcFetchAsync** (the default value), because it provides better performance. In a compiled client application, you will usually want to use **adcFetchBackground**.

See Also

[VBScript Example](#)

[Cancel Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

FilterColumn Property (RDS)

Indicates the column on which to evaluate the filter criteria.

Syntax

```
DataControl.FilterColumn = String
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

String

A **String** value that specifies the column on which to evaluate the filter criteria. The filter criteria are specified in the [FilterCriterion](#) property.

Remarks

The [SortColumn](#), [SortDirection](#), [FilterValue](#), [FilterCriterion](#), and **FilterColumn** properties provide sorting and filtering functionality on the [client-side](#) cache. The sorting functionality orders records by values from one column. The filtering functionality displays a subset of records based on find criteria, while the full [Recordset](#) is maintained in the cache. The [Reset](#) method will execute the criteria and replace the current **Recordset** with an updatable **Recordset**.

See Also

[VBScript Example](#)

[FilterCriterion Property \(RDS\)](#) | [FilterValue Property \(RDS\)](#) | [SortColumn Property \(RDS\)](#) | [SortDirection Property \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

RDS 2.5 API Reference

FilterCriterion Property (RDS)

Indicates the evaluation operator to use in the filter value.

Syntax

```
DataControl.FilterCriterion = String
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

String

A **String** value that specifies the evaluation operator of the [FilterValue](#) to the records. Can be any one of the following: <, <=, >, >=, =, or <>.

Remarks

The [SortColumn](#), [SortDirection](#), [FilterValue](#), **FilterCriterion**, and [FilterColumn](#) properties provide sorting and filtering functionality on the [client-side](#) cache. The sorting functionality orders records by values from one column. The filtering functionality displays a subset of records based on find criteria, while the full [Recordset](#) is maintained in the cache. The [Reset](#) method will execute the criteria and replace the current **Recordset** with an updatable **Recordset**.

The "!=" operator is not valid for **FilterCriterion**; instead, use "<>".

If both the filter and sort properties are set and you call the **Reset** method, the [rowset](#) is first filtered and then it is sorted. For ascending sorts, the null values are at the top; for descending sorts, null values are at the bottom (ascending is default behavior).

See Also

[VBScript Example](#)

[FilterColumn Property \(RDS\)](#) | [FilterValue Property \(RDS\)](#) | [SortColumn Property \(RDS\)](#) | [SortDirection Property \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

FilterValue Property (RDS)

Indicates the value with which to filter records.

Syntax

```
DataControl.FilterValue = String
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

String

A **String** value that represents a data value with which to filter records (for example, 'Programmer' or 125).

Remarks

The [SortColumn](#), [SortDirection](#), **FilterValue**, [FilterCriterion](#), and [FilterColumn](#) properties provide sorting and filtering functionality on the [client-side](#) cache. The sorting functionality orders records by values from one column. The filtering functionality displays a subset of records based on find criteria, while the full [Recordset](#) is maintained in the cache. The [Reset](#) method will execute the criteria and replace the current **Recordset** with an updatable **Recordset**.

Null values result in a type mismatch error.

See Also

[VBScript Example](#)

[FilterColumn Property \(RDS\)](#) | [FilterCriterion Property \(RDS\)](#) | [SortColumn Property \(RDS\)](#) | [SortDirection Property \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

Handler Property (RDS)

Indicates the name of a server-side customization program (handler) that extends the functionality of the [RDS.Server.DataFactory](#), and any parameters used by the *handler*.

Syntax

```
DataControl.Handler = String
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

String

A **String** value that contains the name of the handler and any parameters, all separated by commas (for example, "handlerName, parm1, parm2, . . . , parmN").

Remarks

This property supports [customization](#), a functionality that requires setting the [CursorLocation](#) property to **adUseClient**.

The name of the handler and its parameters, if any, are separated by commas (","). Unpredictable behavior will result if a semicolon (";") appears anywhere within *String*. You can write your own handler, provided it supports the **IDataFactoryHandler** interface.

The name of the default handler is **MSDFMAP.Handler**, and its default parameter is a customization file named **MSDFMAP.INI**. Use this property to invoke alternate customization files created by your server administrator.

The alternative to setting the **Handler** property is to specify a handler and parameters in the [ConnectionString](#) property; that is, "**Handler**=handlerName,parm1,parm2,...;".

See Also

[Visual Basic Example](#)

[DataFactory Customization](#) | [DataFactory Object \(RDS Server\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

InternetTimeout Property (RDS)

Indicates the number of milliseconds to wait before a request times out.

Settings and Return Values

Sets or returns a **Long** value that represents the number of milliseconds before a request will time out.

Remarks

This property applies only to requests sent with the HTTP or HTTPS protocols.

Requests in a three-tier environment can take several minutes to execute. Use this property to specify additional time for long-running requests.

See Also

[Visual Basic Example](#) | [VC++ Example](#) | [VJ++ Example](#)

Applies To: [DataControl Object \(RDS\)](#) | [DataSpace Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

ReadyState Property (RDS)

Indicates the progress of a [DataControl](#) object as it retrieves data into its [Recordset](#) object.

Settings and Return Values

Sets or returns one of the following values.

Value	Description
adcReadyStateLoaded	The current query is still executing and no rows have been fetched. The DataControl object's Recordset is not available for use.
adcReadyStateInteractive	An initial set of rows retrieved by the current query has been stored in the DataControl object's Recordset and are available for use. The remaining rows are still being fetched.
adcReadyStateComplete	All rows retrieved by the current query have been stored in the DataControl object's Recordset and are available for use.
	This state will also exist if an operation aborted due to an error, or if the Recordset object is not initialized.

Note Each [client-side](#) executable file that uses these constants must provide declarations for them. You can cut and paste the constant declarations you want from the file `Adcvbs.inc`, located in the `C:\Program Files\Common Files\System\MSADC` folder.

Remarks

Use the [onReadyStateChange](#) event to monitor changes in the **ReadyState** property during an [asynchronous](#) query operation. This is more efficient than

periodically checking the value of the property.

If an error occurs during an [asynchronous](#) operation, the **ReadyState** property changes to **adReadyStateComplete**, the [State](#) property changes from **adStateExecuting** to **adStateClosed**, and the **Recordset** object [Value](#) property remains *Nothing*.

See Also

[VBScript Example](#)

[Cancel Method \(RDS\)](#) | [ExecuteOptions Property \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Recordset, SourceRecordset Properties (RDS)

Indicates the **Recordset** object returned from a custom [business object](#).

Syntax

```
DataControl.SourceRecordset = Recordset  
Recordset = DataControl.Recordset
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

Recordset

An object variable that represents a **Recordset** object.

Remarks

You can set the **SourceRecordset** property to a [Recordset](#) returned from a custom business object.

These properties allow an application to handle the binding process by means of a custom process. They receive a [rowset](#) wrapped in a **Recordset** so that you can interact directly with the **Recordset**, performing actions such as setting properties or iterating through the **Recordset**.

You can set the **SourceRecordset** property or read the **Recordset** property at run time in scripting code.

SourceRecordset is a write-only property, in contrast to the **Recordset** property, which is a read-only property.

See Also

[VBScript Example](#)

[CreateRecordset Method \(RDS\)](#) | [Query Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Server Property (RDS)

Indicates the Internet Information Services (IIS) name and communication protocol.

You can set the **Server** property at design time in the [RDS.DataControl](#) object's OBJECT tags, or at run time in scripting code.

Syntax

Protocol	Design-time syntax
HTTP	<code><PARAM NAME="Server" VALUE="http://awebsrvr:port"></code>
HTTPS	<code><PARAM NAME="Server" VALUE="https://awebsrvr:port"></code>
DCOM	<code><PARAM NAME="Server" VALUE="computername"></code>
In-process	<code><PARAM NAME="Server" VALUE=""></code>

Protocol	Run-time syntax
HTTP	<code><i>DataControl</i>.Server="http://awebsrvr:port"</code>
HTTPS	<code><i>DataControl</i>.Server="https://awebsrvr:port"</code>
DCOM	<code><i>DataControl</i>.Server="computername"</code>
In-process	<code><i>DataControl</i>.Server=""</code>

Parameters

awebsrvr or *computername*

A **String** value that contains an Internet or intranet path, or computer name, if the server is on a remote computer; or, an empty string if the server is on the local computer.

port

Optional. A port that is used to connect to an IIS server. The port number is set in Internet Explorer (on the **View** menu, click **Options**, and then select the **Connection** tab) or in IIS.

DataControl

An [object variable](#) that represents an **RDS.DataControl** object.

Remarks

The server is the location where the **RDS.DataControl** request (that is, a query or update) is processed. By default, all requests are processed by the [RDS.Server.DataFactory](#) object, [MSDFMAP.Handler](#) component, and [MSDFMAP.INI](#) file on the specified server. Remember that when changing servers to reconcile settings in the old and new **MSDFMAP.INI** files. Incompatibilities may cause requests that succeed on one server to fail on another. If the Server property is set to "", these objects will be used on the local machine.

See Also

[VBScript Example](#)

[Connect Property \(RDS\)](#) | [SQL Property \(RDS\)](#) | [SubmitChanges Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

SortColumn Property (RDS)

Indicates by which column to sort the records.

Syntax

```
DataControl.SortColumn = String
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

String

A **String** value that represents the name or alias of the column by which to sort the records.

Remarks

The **SortColumn**, [SortDirection](#), [FilterValue](#), [FilterCriterion](#), and [FilterColumn](#) properties provide sorting and filtering functionality on the client-side cache. The sorting functionality orders records by values from one column. The filtering functionality displays a subset of records based on find criteria, while the full [Recordset](#) is maintained in the cache. The [Reset](#) method will execute the criteria and replace the current **Recordset** with an updatable **Recordset**.

To sort on a **Recordset**, you must first save any pending changes. If you are using the **RDS.DataControl**, you can use the [SubmitChanges](#) method. For example, if your **RDS.DataControl** is named ADC1, your code would be `ADC1.SubmitChanges`. If you are using an ADO **Recordset**, you can use its [UpdateBatch](#) method. Using **UpdateBatch** is the recommended method for **Recordset** objects created with the [CreateRecordset](#) method. For example, your code could be `myRS.UpdateBatch` or `ADC1.Recordset.UpdateBatch`.

See Also

[VBScript Example](#)

[Sort Property](#) | [SortDirection Property \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

SortDirection Property (RDS)

Indicates whether a sort order is ascending or descending.

Syntax

```
DataControl.SortDirection = value
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

Value

A **Boolean** value that, when set to **True**, indicates the sort direction is ascending. **False** indicates descending order.

Remarks

The [SortColumn](#), **SortDirection**, [FilterValue](#), [FilterCriterion](#), and [FilterColumn](#) properties provide sorting and filtering functionality on the client-side cache. The sorting functionality orders records by values from one column. The filtering functionality displays a subset of records based on find criteria, while the full [Recordset](#) is maintained in the cache. The **Reset** method will execute the criteria and replace the current **Recordset** with an updatable **Recordset**.

See Also

[VBScript Example](#)

[Sort Property](#) | [SortColumn Property \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

RDS 2.5 API Reference

SQL Property (RDS)

Indicates the query string used to retrieve the [Recordset](#).

You can set the **SQL** property at design time in the [RDS.DataControl](#) object's OBJECT tags, or at run time in scripting code.

Syntax

Design time: `<PARAM NAME="SQL" VALUE="QueryString">`

Run time: `DataControl.SQL = "QueryString"`

Parameters

QueryString

A **String** value that contains a valid SQL data request.

DataControl

An [object variable](#) that represents an **RDS.DataControl** object.

Remarks

In general, this is an SQL statement (using the dialect of the database server), such as "Select * from NewTitles". To ensure that records are matched and updated accurately, an updatable query must contain a field other than a Long Binary field or a computed field.

The **SQL** property is optional if a custom server-side [business object](#) retrieves the data for the client.

See Also

[VBScript Example](#)

[Connect Property \(RDS\)](#) | [Query Method \(RDS\)](#) | [Refresh Method \(RDS\)](#) | [SubmitChanges Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

URL Property (RDS)

Indicates a string that contains a [relative](#) or [absolute URL](#).

You can set the **URL** property at design time in the [DataControl](#) object's OBJECT tag, or at run time in scripting code.

Syntax

Design time: `<PARAM NAME="URL" VALUE="Server">`

Run time: `DataControl.URL="Server"`

Parameters

Server

A **String** value that contains a valid URL.

DataControl

An [object variable](#) that represents a **DataControl** object.

Remarks

Typically, the URL identifies an Active Server Page (.asp) file that can produce and return a [Recordset](#). Therefore, the user can obtain a **Recordset** without having to invoke the server-side [DataFactory](#) object, or program a custom business object.

If the **URL** property has been set, [SubmitChanges](#) will submit changes to the location specified by the URL.

See Also

[VBScript Example](#)

Applies To: [DataControl Object \(RDS\)](#)

RDS 2.5 API Reference

RDS Methods

Cancel (RDS)	Cancels execution of a pending, asynchronous method call.
CancelUpdate (RDS)	Cancels any changes made to the current or new row of a Recordset object.
ConvertToString (RDS)	Converts a Recordset to a MIME string that represents the recordset data.
CreateObject (RDS)	Creates the proxy for the target business object and returns a pointer to it.
CreateRecordset (RDS)	Creates an empty, disconnected Recordset .
MoveFirst, MoveLast, MoveNext, MovePrevious (RDS)	Moves to the first, last, next, or previous record in a specified Recordset object.
Query (RDS)	Uses a valid SQL query string to return a Recordset .
Refresh (RDS)	Requeries the data source specified in the Connect property and updates the query results.
Reset (RDS)	Executes the sort or filter on a client-side Recordset , based on the specified sort and filter properties.
SubmitChanges (RDS)	Submits pending changes of the locally cached and updatable Recordset to the data source specified in the Connect property.

RDS 2.5 API Reference

Cancel Method (RDS)

Cancels execution of a pending, [asynchronous](#) method call.

Syntax

```
RDS.DataControl.Cancel
```

Remarks

When you call **Cancel**, [ReadyState](#) is automatically set to **adcReadyStateLoaded**, and the [Recordset](#) will be empty.

See Also

[VBScript Example](#)

[Cancel Method](#) | [CancelBatch Method](#) | [CancelUpdate Method](#) | [CancelUpdate Method \(RDS\)](#) | [ExecuteOptions Property \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

CancelUpdate Method (RDS)

Cancels any changes made to the current or new row of a [Recordset](#) object.

Syntax

DataControl.**CancelUpdate**

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

Remarks

The Cursor Service for OLE DB keeps both a copy of the original values and a cache of changes. When you call **CancelUpdate**, the cache of changes is reset to empty, and any bound controls are refreshed with the original data.

See Also

[VBScript Example](#)

[Address Book Command Buttons](#) | [Cancel Method](#) | [Cancel Method \(RDS\)](#) | [CancelBatch Method](#) | [CancelUpdate Method](#) | [Refresh Method \(RDS\)](#) | [SubmitChanges Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

ConvertToString Method (RDS)

Converts a [Recordset](#) to a [MIME](#) string that represents the recordset data.

Syntax

```
DataFactory.ConvertToString(Recordset)
```

Parameters

DataFactory

An [object variable](#) that represents an [RDS.Server.DataFactory](#) object.

Recordset

An object variable that represents a **Recordset** object.

Remarks

With .asp files, use **ConvertToString** to embed the **Recordset** in an HTML page generated on the server to transport it to a client computer.

ConvertToString first loads the **Recordset** into the Cursor Service tables, and then generates a stream in MIME format.

On the client, Remote Data Service can convert the MIME string back into a fully functioning **Recordset**. It works well for handling fewer than 400 rows of data with no more than 1024 bytes width per row. You shouldn't use it for streaming BLOB data and large result sets over HTTP. No wire compression is performed on the string, so very large data sets will take considerable time to transport over HTTP when compared to the wire-optimized tablegram format defined and deployed by Remote Data Service as its native transport protocol format.

Note If you are using Active Server Pages to embed the resulting MIME string in a client HTML page, be aware that versions of VBScript earlier than version 2.0 limit the string's size to 32K. If this limit is exceeded, an error is returned. Keep the query scope relatively small when using MIME

embedding via .asp files. To fix this, download the latest version of VBScript from the [Microsoft Windows Script Technologies Web site](#).

See Also

[Visual Basic Example](#) | [VBScript Example](#)

Applies To: [DataFactory Object \(RDS Server\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

CreateObject Method (RDS)

Creates the proxy for the target [business object](#) and returns a pointer to it. The [proxy](#) packages and [marshals](#) data to the server-side [stub](#) for communications with the business object to send requests and data over the Internet. For in-process component objects, no proxies are used, just a pointer to the object is provided.

Syntax

Remote Data Service supports the following protocols: HTTP, HTTPS (HTTP over Secure Socket Layer), [DCOM](#), and in-process.

Protocol	Syntax
HTTP	<code>Set object = DataSpace.CreateObject("ProgId", "http://awebsr</code>
HTTPS	<code>Set object = DataSpace.CreateObject("ProgId", "https://awebsr</code>
DCOM	<code>Set object = DataSpace.CreateObject("ProgId", "computername'</code>
In-process	<code>Set object = DataSpace.CreateObject("ProgId", "")</code>

Parameters

Object

An [object variable](#) that evaluates to an object that is the type specified in *ProgID*.

DataSpace

An object variable that represents an [RDS.DataSpace](#) object used to create an instance of the new object.

ProgID

A **String** value that contains the programmatic identifier specifying a server-side [business object](#) that implements your application's [business](#)

[rules](#).

awebsrvr or *computername*

A **String** value that represents a URL identifying the Internet Information Services (IIS) Web server where an instance of the server business object is created.

Remarks

The *HTTP protocol* is the standard Web protocol; *HTTPS* is a secure Web protocol. Use the *DCOM protocol* when running a local-area network without HTTP. The *in-process* protocol is a local dynamic-link library (DLL); it does not use a network.

See Also

[DataFactory Object, Query Method, and CreateObject Method Example \(VBScript\)](#) | [DataSpace Object and CreateObject Method Example \(VBScript\)](#)

[CreateRecordset Method \(RDS\)](#)

Applies To: [DataSpace Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

CreateRecordset Method (RDS)

Creates an empty, disconnected [Recordset](#).

Syntax

```
object.CreateRecordset(ColumnInfos)
```

Parameters

Object

An [object variable](#) that represents an [RDS.Server.DataFactory](#) or [RDS.DataControl](#) object.

ColumnsInfos

A **Variant** array of attributes that defines each column in the **Recordset** created. Each column definition contains an array of four required attributes and one optional attribute.

Attribute	Description
Name	Name of the column header.
Type	Integer of the data type.
Size	Integer of the width in characters, regardless of data type.
Nullability	Boolean value.
Scale (Optional)	This optional attribute defines the scale for numeric fields. If this value is not specified, numeric values will be truncated to a scale of three. Precision is not affected, but the number of digits following the decimal point will be truncated to three.

The set of column arrays is then grouped into an array, which defines the **Recordset**.

Remarks

The server-side [business object](#) can populate the resulting **Recordset** with data from a non-OLE DB [data provider](#), such as an operating system file containing stock quotes.

The following table lists the [DataTypeEnum](#) values supported by the **CreateRecordset** method. The number listed is the reference number used to define fields.

Each of the data types is either fixed length or variable length. Fixed-length types should be defined with a size of -1, because the size is predetermined and a size definition is still required. Variable-length data types allow a size from 1 to 32767.

For some of the variable data types, the type may be coerced to the type noted in the Substitution column. You won't see the substitutions until after the **Recordset** is created and filled. Then you can check for the actual data type, if necessary.

Length	Constant	Number	Substitution
Fixed	adTinyInt	16	
Fixed	adSmallInt	2	
Fixed	adInteger	3	
Fixed	adBigInt	20	
Fixed	adUnsignedTinyInt	17	
Fixed	adUnsignedSmallInt	18	
Fixed	adUnsignedInt	19	
Fixed	adUnsignedBigInt	21	
Fixed	adSingle	4	
Fixed	adDouble	5	
Fixed	adCurrency	6	
Fixed	adDecimal	14	
Fixed	adNumeric	131	
Fixed	adBoolean	11	
Fixed	adError	10	
Fixed	adGuid	72	
Fixed	adDate	7	

Fixed	adDBDate	133	
Fixed	adDBTime	134	
Fixed	adDBTimestamp	135	7
Variable	adBSTR	8	130
Variable	adChar	129	200
Variable	adVarChar	200	
Variable	adLongVarChar	201	200
Variable	adWChar	130	
Variable	adVarWChar	202	130
Variable	adLongVarWChar	203	130
Variable	adBinary	128	
Variable	adVarBinary	204	
Variable	adLongVarBinary	205	204

See Also

[Visual Basic Example](#) | [VBScript Example](#)

[CreateObject Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#) | [DataFactory Object \(RDSServer\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

MoveFirst, MoveLast, MoveNext, and MovePrevious Methods (RDS)

Moves to the first, last, next, or previous record in a specified [Recordset](#) object.

Syntax

```
DataControl.Recordset.{MoveFirst | MoveLast | MoveNext | MovePrevious}
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

Remarks

You can use the **Move** methods with the **RDS.DataControl** object to navigate through the data records in the data-bound controls on a Web page. For example, suppose you display a **Recordset** in a grid by binding to an **RDS.DataControl** object. You can then include First, Last, Next, and Previous buttons that users can click to move to the first, last, next, or previous record in the displayed **Recordset**. You do this by calling the **MoveFirst**, **MoveLast**, **MoveNext**, and **MovePrevious** methods of the **RDS.DataControl** object in the onClick procedures for the First, Last, Next, and Previous buttons, respectively. The [Address Book example](#) shows how to do this.

See Also

[Move Method](#) | [MoveFirst, MoveLast, MoveNext, and MovePrevious Methods](#) | [MoveRecord Method](#)

Applies To: [DataControl Object \(RDS\)](#)

RDS 2.5 API Reference

Query Method (RDS)

Uses a valid SQL query string to return a [Recordset](#).

Syntax

```
Set Recordset = DataFactory.Query(Connection, Query)
```

Parameters

Recordset

An [object variable](#) that represents a **Recordset** object.

DataFactory

An object variable that represents an [RDS.Server.DataFactory](#) object.

Connection

A **String** value that contains the server connection information. This is similar to the [Connect](#) property.

Query

A **String** that contains the SQL query.

Remarks

The query should use the SQL dialect of the database server. A result status is returned if there is an error with the query that was executed. The **Query** method doesn't perform any syntax checking on the **Query** string.

See Also

[VBScript Example](#)

Applies To: [DataFactory Object \(RDS.Server\)](#)

RDS 2.5 API Reference

Refresh Method (RDS)

Requeries the data source specified in the [Connect](#) property and updates the query results.

Syntax

DataControl.**Refresh**

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

Remarks

You must set the [Connect](#), [Server](#), and [SQL](#) properties before you use the **Refresh** method. All data-bound controls on the form associated with an **RDS.DataControl** object will reflect the new set of records. Any pre-existing [Recordset](#) object is released, and any unsaved changes are discarded. The **Refresh** method automatically makes the first record the current record.

It's a good idea to call the **Refresh** method periodically when you work with data. If you retrieve data, and then leave it on your client machine for a while, it is likely to become out of date. It's possible that any changes you make will fail, because someone else might have changed the record and submitted changes before you.

See Also

[Visual Basic Example](#) | [VBScript Example](#)

[Address Book Command Buttons](#) | [CancelUpdate Method \(RDS\)](#) | [Refresh Method](#) | [SubmitChanges Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

Reset Method (RDS)

Executes the sort or filter on a [client-side Recordset](#) based on the specified sort and filter properties.

Syntax

```
DataControl.Reset(value)
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

value

Optional. A **Boolean** value that is **True** (default) if you want to filter on the current "filtered" [rowset](#). **False** indicates that you filter on the original rowset, removing any previous filter options.

Remarks

The [SortColumn](#), [SortDirection](#), [FilterValue](#), [FilterCriterion](#), and [FilterColumn](#) properties provide sorting and filtering functionality on the client-side cache. The sorting functionality orders records by values from one column. The filtering functionality displays a subset of records based on a find criteria, while the full [Recordset](#) is maintained in the cache. The **Reset** method will execute the criteria and replace the current **Recordset** with an updatable **Recordset**.

If there are changes to the original data that haven't yet been submitted, the **Reset** method will fail. First, use the [SubmitChanges](#) method to save any changes in a read/write **Recordset**, and then use the **Reset** method to sort or filter the records.

If you want to perform more than one filter on your rowset, you can use the optional *Boolean* argument with the **Reset** method. The following example shows how to do this:

```
ADC.SQL = "Select au_lname from authors"
ADC.Refresh      ' Get the new rowset.

ADC.FilterColumn = "au_lname"
ADC.FilterCriterion = "<"
ADC.FilterValue = "'M'"
ADC.Reset      ' Rowset now has all Last Names < "M".

ADC.FilterCriterion = ">"
ADC.FilterValue = "'F'"
' Passing True is not necessary, because it is the
' default filter on the current "filtered" rowset.
ADC.Reset(TRUE)      ' Rowset now has all Last
                    ' Names < "M" and > "F".

ADC.FilterCriterion = ">"
ADC.FilterValue = "'T'"
' Filter on the original rowset, throwing out the
' previous filter options.
ADC.Reset(FALSE)    ' Rowset now has all Last Names > "T".
```

See Also

[VBScript Example](#)

[SubmitChanges Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

SubmitChanges Method (RDS)

Submits pending changes of the locally cached and updatable [Recordset](#) to the data source specified in the [Connect](#) property or the [URL](#) property.

Syntax

```
DataControl.SubmitChanges  
DataFactory.SubmitChanges Connection, Recordset
```

Parameters

DataControl

An [object variable](#) that represents an [RDS.DataControl](#) object.

DataFactory

An object variable that represents an [RDSServer.DataFactory](#) object.

Connection

A **String** value that represents the connection created with the **RDS.DataControl** object's **Connect** property.

Recordset

An object variable that represents a **Recordset** object.

Remarks

The [Connect](#), [Server](#), and [SQL](#) properties must be set before you can use the **SubmitChanges** method with the **RDS.DataControl** object.

If you call the [CancelUpdate](#) method after you have called **SubmitChanges** for the same **Recordset** object, the **CancelUpdate** call fails because the changes have already been committed.

Only the changed records are sent for modification, and either all of the changes succeed or all of them fail together.

You can use **SubmitChanges** only with the *default* **RDSServer.DataFactory** object. Custom [business objects](#) can't use this method.

If the **URL** property has been set, **SubmitChanges** will submit changes to the location specified by the URL.

See Also

[VBScript Example](#)

[Address Book Command Buttons](#) | [CancelUpdate Method \(RDS\)](#) | [Refresh Method \(RDS\)](#)

Applies To: [DataControl Object \(RDS\)](#) | [DataFactory Object \(RDSServer\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

RDS Events

[onError \(RDS\)](#)

Called whenever an error occurs during an operation.

[onReadyStateChange \(RDS\)](#)

Called whenever the value of the **ReadyState** property changes.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

onError Event (RDS)

The **onError** event is called whenever an error occurs during an operation.

Syntax

```
onError SCode, Description, Source, CancelDisplay
```

Parameters

SCode

An integer that indicates the status code of the error.

Description

A **String** that indicates a description of the error.

Source

A **String** that indicates the query or command that caused the error.

CancelDisplay

A **Boolean** value, which if set to **True**, that prevents the error from being displayed in a dialog box.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

onReadyStateChange Event (RDS)

The **onReadyStateChange** event is called whenever the value of the [ReadyState](#) property changes.

Syntax

onReadyStateChange

Parameters

None.

Remarks

The **ReadyState** property reflects the progress of an [RDS.DataControl](#) object as it asynchronously retrieves data into its [Recordset](#) object. Use the **onReadyStateChange** event to monitor changes in the **ReadyState** property whenever they occur. This is more efficient than periodically checking the property's value.

See Also

[Visual C++ Example](#)

[ADO Event Handler Summary](#)

Applies To: [DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

RDS Code Examples

Use the following code examples to learn how to use the RDS objects, methods, properties, and events.

Note Paste the entire code example into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

- [RDS Code Examples in Microsoft Visual Basic](#)
- [RDS Code Examples in Microsoft Visual Basic Scripting Edition](#)
- [RDS Code Examples in Microsoft Visual C++](#)
- [RDS Code Examples in Microsoft Visual J++](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

RDS Code Examples in Microsoft Visual Basic

Use the following code examples to learn how to use RDS properties when writing in Visual Basic.

Note Paste the entire code example, from Sub to End Sub, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

Properties

- [Handler Property Example](#)
- [InternetTimeout Property Example](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Handler Property Example (VB)

This example demonstrates the [RDS DataControl](#) object [Handler](#) property. (See [DataFactory Customization](#) for more details.)

Assume that the following sections in the parameter file, Msdfmap.ini, are located on the server:

```
[connect AuthorDataBase]
Access=ReadWrite
Connect="DSN=Pubs"
[sql AuthorById]
SQL="SELECT * FROM Authors WHERE au_id = ?"
```

Your code looks like the following. The command assigned to the [SQL](#) property will match the **AuthorById** identifier and will retrieve a row for author Michael O'Leary. The **DataControl** object **Recordset** property is assigned to a disconnected [Recordset](#) object purely as a coding convenience.

```
'BeginHandlerVB

Public Sub Main()

On Error GoTo ErrorHandler

Dim dc As New DataControl

Dim rst As ADODB.Recordset

dc.Handler = "MSDFMAP.Handler"

dc.ExecuteOptions = 1

dc.FetchOptions = 1

dc.Server = "http://MyServer"

dc.Connect = "Data Source=AuthorDataBase"

dc.SQL = "AuthorById('267-41-2394')"
```

dc.Refresh 'Retrieve the record

```
Set rst = dc.Recordset 'Use another Recordset as a convenience
Debug.Print "Author is '" & rst!au_fname & " " & rst!au_lname & "'"
' clean up
If rst.State = adStateOpen Then rst.Close
Set rst = Nothing
Set dc = Nothing
Exit Sub
ErrorHandler:
' clean up
If Not rst Is Nothing Then
If rst.State = adStateOpen Then rst.Close
End If
Set rst = Nothing
Set dc = Nothing
If Err <> 0 Then
MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndHandlerVB
```

See Also

[DataControl Object \(RDS\)](#) | [Handler Property \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

InternetTimeout Property Example (VB)

This example demonstrates the [InternetTimeout](#) property, which exists on the [DataControl](#) and [DataSpace](#) objects. This example uses the **DataControl** object and sets the timeout to 20 seconds.

```
'BeginInternetTimeoutVB

Public Sub Main()

On Error GoTo ErrorHandler

Dim dc As RDS.DataControl

Dim rst As ADODB.Recordset

Set dc = New RDS.DataControl

dc.Server = "http://MyServer"

dc.ExecuteOptions = 1

dc.FetchOptions = 1

dc.Connect = "Provider='sqloledb';Data Source='MySqlServer';" & _
"Initial Catalog='Pubs';Integrated Security='SSPI';"

dc.SQL = "SELECT * FROM Authors"

' Wait at least 20 seconds

dc.InternetTimeout = 200

dc.Refresh

' Use another Recordset as a convenience

Set rst = dc.Recordset
```

```
Do While Not rst.EOF
Debug.Print rst!au_fname & " " & rst!au_lname
rst.MoveNext
Loop
If rst.State = adStateOpen Then rst.Close
Set rst = Nothing
Set dc = Nothing
Exit Sub
ErrorHandler:
' clean up
If Not rst Is Nothing Then
If rst.State = adStateOpen Then rst.Close
End If
Set rst = Nothing
Set dc = Nothing
If Err <> 0 Then
MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
'EndInternetTimeoutVB
```

See Also

[DataControl Object \(RDS\)](#) | [DataSpace Object \(RDS\)](#) | [InternetTimeout Property \(RDS\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

RDS Code Examples in Microsoft Visual Basic Scripting Edition

Use the following code examples to learn how to use the RDS objects, methods, and properties when writing in Microsoft Visual Basic Scripting Edition (VBScript).

Note Paste the entire code example into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

Objects

- [DataControl Object Example](#)
- [DataSpace Object Example](#)
- [DataFactory Object Example](#)

Methods

- [Cancel Method Example](#)
- [CancelUpdate Method Example](#)
- [ConvertToString Method Example](#)
- [CreateObject Method](#)
- [CreateRecordset Method Example](#)
- [Query Method Example](#)
- [Refresh Method Example](#)
- [Reset Method Example](#)
- [SubmitChanges Method Example](#)

Properties

- [Connect Property Example](#)
- [ExecuteOptions and FetchOptions Properties Example](#)
- [FilterColumn, FilterCriterion, FilterValue, SortColumn, and SortDirection](#)

Properties Example

- [ReadyState Property Example](#)
- [Recordset and SourceRecordset Properties Example](#)
- [Server Property Example](#)
- [SQL Property Example](#)
- [URL Property Example](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Cancel Method Example (VBScript)

The following example shows how to read the [Cancel](#) method at run time. Cut and paste the following code to Notepad or another text editor and save it as **CancelVBS.asp**. You can view the result in any [client](#) browser.

```
<!-- BeginCancelVBS -->

<Script Language="VBScript">

<!--

Sub cmdCancelAsync_OnClick

' Terminates currently running AsyncExecute,
' ReadyState property set to adcReadyStateLoaded,
' Recordset set to Nothing

ADC.Cancel

End Sub

Sub cmdRefreshTable_OnClick

ADC.Refresh

End Sub

-->

</Script>

<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
ID="ADC">

.

<PARAM NAME="SQL" VALUE="Select FirstName, LastName from
Employees">

<PARAM NAME="CONNECT" VALUE="Provider='sqloledb';Integrated
```

```
Security='SSPI';Initial Catalog='Northwind'">
<PARAM NAME="Server"
VALUE="http://<%=Request.ServerVariables("SERVER_NAME")%>">
.
</OBJECT>
<TABLE DATASRC=#ADC>
<TBODY>
<TR>
<TD><SPAN DATAFLD="FirstName"></SPAN></TD>
<TD><SPAN DATAFLD="LastName"></SPAN></TD>
</TR>
</TBODY>
</TABLE>
<FORM>
<INPUT type="button" value="Refresh" id=cmdRefreshTable
name=cmdRefreshTable>
<INPUT type="button" value="Cancel" id=cmdCancelAsync
name=cmdCancelAsync>
</FORM>
<!-- EndCancelVBS -->
```

See Also

[Cancel Method](#)

RDS 2.5 API Reference

CancelUpdate Method Example (VBScript)

To test this example, cut and paste this code between the <Body> and </Body> tags in a normal HTML document and name it **CancelUpdateVBS.asp**. ASP script will identify your internet server. You will need to edit the name of the server to reflect your own setup. Simply change the value in the connect string from MyServer to the name of your SQL Server installation.

```
<!-- BeginCancelUpdateVBS -->

<%@Language=VBScript%>

<%'Option Explicit%>

<% 'use the following META tag instead of adovbs.inc%>

<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-
00AA006D2EA4" -->

<HTML>

<HEAD>

<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">

<TITLE></TITLE>

</HEAD>

<BODY>

<CENTER>

<H1>Remote Data Service</H1>

<H2>SubmitChanges and CancelUpdate Methods</H2>

<% ' to integrate/test this code replace the Server property value
and
```

' the Data Source value in the Connect property with appropriate values%>

<HR>

<OBJECT ID=RDS classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" HEIGHT=1 WIDTH=1></OBJECT>

<SCRIPT Language="VBScript">

'set RDS properties for control just created

RDS.Server = "http://<%=Request.ServerVariables("SERVER_NAME")%>"

RDS.SQL = "Select * from Employees"

RDS.Connect = "Provider='sqloledb';Integrated Security='SSPI';Initial Catalog='Northwind';"

RDS.Refresh

</SCRIPT>

<TABLE DATASRC=#RDS>

<THEAD>

<TR ID="ColHeaders">

<TH>ID</TH>

<TH>FName</TH>

<TH>LName</TH>

<TH>Title</TH>

<TH>Hire Date</TH>

<TH>Birth Date</TH>

<TH>Extension</TH>

<TH>Home Phone</TH>

</TR>

```
</THEAD>

<TBODY>

<TR>

<TD> <INPUT DATAFLD="EmployeeID" size=4> </TD>

<TD> <INPUT DATAFLD="FirstName" size=10> </TD>

<TD> <INPUT DATAFLD="LastName" size=10> </TD>

<TD> <INPUT DATAFLD="Title" size=10> </TD>

<TD> <INPUT DATAFLD="HireDate" size=10> </TD>

<TD> <INPUT DATAFLD="BirthDate" size=10> </TD>

<TD> <INPUT DATAFLD="Extension" size=10> </TD>

<TD> <INPUT DATAFLD="HomePhone" size=8> </TD>

</TR>

</TBODY>

</TABLE>

<HR>

<INPUT TYPE=button NAME="SubmitChange" VALUE="Submit Changes">

<INPUT TYPE=button NAME="CancelChange" VALUE="Cancel Update">

<BR>

<H4>Alter a current entry on the grid. Move off that Row. <BR>

Submit the Changes to your DBMS or cancel the updates. </H4>

</CENTER>

<SCRIPT Language="VBScript">

Sub SubmitChange_OnClick

msgbox "Changes will be made"
```

```
RDS.SubmitChanges
RDS.Refresh
End Sub
Sub CancelChange_OnClick
msgbox "Changes will be cancelled"
RDS.CancelUpdate
RDS.Refresh
End Sub
-->
</SCRIPT>
</BODY>
</HTML>
<!-- EndCancelUpdateVBS -->
```

See Also

[CancelUpdate Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Connect Property Example (VBScript)

This code shows how to set the [Connect](#) property at design time:

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID="ADC"
.
  <PARAM NAME="SQL" VALUE="Select * from Sales">
  <PARAM NAME="CONNECT" VALUE="Provider=SQLOLEDB;Integrated Security=SSPI;Server=localhost;">
  <PARAM NAME="Server" VALUE="http://MyWebServer">
.
</OBJECT>
```

The following example shows how to set the **Connect** property at run time in VBScript code.

To test this example, cut and paste this code between the <Body> and </Body> tags in a normal HTML document and name it **ConnectVBS.asp**. ASP script will identify your server.

```
<!-- BeginConnectVBS -->

<%@ Language=VBScript %>

<HTML>

<HEAD>

<title>ADO Connect Property</title>

<% ' local style sheet used for display%>

<STYLE>

<!--

BODY {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;
```

```
COLOR:black;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</STYLE>
</HEAD>
<BODY>
<h1>ADO Connect Property (RDS)</h1>
<HR>
<H3>Set Connect Property at Run Time</H3>
<% ' RDS.DataControl with no parameters set at design time %>
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID=RDS
HEIGHT=1 WIDTH=1></OBJECT>
<% ' Bind table to control for data display %>
<TABLE DATASRC=#RDS>
<TBODY>
<TR class="tbody">
<TD><SPAN DATAFLD="FirstName"></SPAN></TD>
<TD><SPAN DATAFLD="LastName"></SPAN></TD>
```

```

</TR>

</TBODY>

</TABLE>

<FORM name="frmInput">

SERVER: <INPUT Name="txtServer" Size="103"
Value="http://<%=Request.ServerVariables("SERVER_NAME")%>"><BR>

DATA SOURCE: <INPUT Name="txtDataSource" Size="93" Value="
<%=Request.ServerVariables("SERVER_NAME")%>"><BR>

CONNECT: <INPUT Name="txtConnect" Size="100"><BR>

SQL: <INPUT Name="txtSQL" Size="110" Value="Select FirstName,
LastName from Employees">

<BR>

<INPUT TYPE=BUTTON NAME="Run" VALUE="Run">

<h4>

To make data grid appear, click 'Run' to see the connect string in
text box above.

</h4>

</FORM>

<Script Language="VBScript">

' Set parameters of RDS.DataControl at Run Time

Sub Run_OnClick

Dim Cnxn

' build connection string

Cnxn = "Provider='sqloledb';"

Cnxn = Cnxn & "Data Source="

Cnxn = Cnxn & document.frmInput.txtDataSource.value & ";"

```

```
Cnxn = Cnxn & "Initial Catalog='Northwind';"  
Cnxn = Cnxn & "Integrated Security='SSPI';"  
' assign the value  
document.frmInput.txtConnect.value = Cnxn  
MsgBox "Here we go!"  
' set RDS properties  
RDS.Server = document.frmInput.txtServer.value  
RDS.SQL = document.frmInput.txtSQL.value  
RDS.Connect = document.frmInput.txtConnect.value  
RDS.Refresh  
End Sub  
</Script>  
</BODY>  
</HTML>  
<!-- EndConnectVBS -->
```

See Also

[Connect Property \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

ConvertToString Method Example (VBScript)

The following example shows how to convert a **Recordset** into a MIME-encoded string using the **RDS.Server.DataFactory ConvertToString** method. It then shows how the string can be converted back into a **Recordset**. Cut and paste the following code to Notepad or another text editor and save it as **ConvertToString.htm**.

```
<!-- BeginConvertToStringVBS -->

<HTML>

<HEAD><TITLE>ConvertToString Example</TITLE><HEAD>

<BODY>

<SCRIPT LANGUAGE=VBSCRIPT>

Sub ConvertToStringX()

Dim objRs, objDF, strServer, vString

Const adcExecSync = 1

Const adcFetchUpFront = 1

' Replace value below with your server name to use without ASP.
strServer = "http://<%=Request.ServerVariables("SERVER_NAME")%>"

Set objDF = RDS1.CreateObject("RDS.Server.DataFactory", strServer)
Set objRs = objDF.Query(txtConnect.Value,txtQueryRecordset.Value)

' convert Recordset to MIME encoded string
vString = objDF.ConvertToString(objRs)

' display MIME string for demo purposes
```

```
txtRS.value = vString
' convert MIME string back to useable ADO Recordset
' using RDS.DataControl
RDC1.SQL = vString
RDC1.ExecuteOptions = adcExecSync
RDC1.FetchOptions = adcFetchUpFront
RDC1.Refresh
MsgBox "RecordCount = " & RDC1.Recordset.RecordCount
End Sub
</SCRIPT>

Connect String:

<INPUT TYPE=Text NAME=txtConnect SIZE=50
VALUE="Provider=sqloledb;Initial Catalog=pubs;Integrated
Security='SSPI';">

<BR>

Query:

<INPUT TYPE=Text NAME=txtQueryRecordset SIZE=50
VALUE="select * from authors">

<BR>

<INPUT TYPE=Button VALUE="ConvertToString"
OnClick="ConvertToStringX()">

<BR>

MIME Encoded RS: <BR>

<TEXTAREA NAME=txtRS ROWS=15 COLS=50 WRAP=virtual></TEXTAREA>

<!-- RDS.DataSpace ID RDS1 -->
```

```
<OBJECT ID="RDS1" WIDTH=1 HEIGHT=1
CLASSID="CLSID:BD96C556-65A3-11D0-983A-00C04FC29E36">
</OBJECT>
<!-- RDS.DataControl ID RDC1 -->
<OBJECT ID="RDC1" WIDTH=1 HEIGHT=1
CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33">
</OBJECT>
</BODY>
</HTML>
<!-- EndConvertToStringVBS -->
```

See Also

[ConvertToString Method \(RDS\)](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

CreateRecordset Method Example (VBScript)

This code example creates a [Recordset](#) on the server side. It has two columns with four rows each. Cut and paste the following code to Notepad or another text editor and save it as **CreateRecordsetVBS.asp**.

```
<!-- BeginCreateRecordsetVBS -->

<%@ Language=VBScript %>

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>CreateRecordset Method Example (VBScript)</title>

<style>

<!--

body {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;

COLOR:black;

}

.thead {

background-color: #008080;

font-family: 'Verdana','Arial','Helvetica',sans-serif;
```

```
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</style>
</head>
<body>
<OBJECT classid=clsid:BD96C556-65A3-11D0-983A-00C04FC29E33 height=1
id=DC1 width=1>
</OBJECT>
<h1>CreateRecordset Method Example (VBScript)</h1>
<script language = "vbscript">
```

```
' use the RDS.DataControl to create an empty recordset;
' takes an array of variants where every element is itself another
' array of variants, one for every column required in the recordset
' the elements of the inner array are the column's
' name, type, size, and nullability
```

```
Sub GetRS()
```

```
Dim Record(2)
```

```
Dim Field1(3)
```

```
Dim Field2(3)
```

```
Dim Field3(3)
```

```
' for each field, specify the name type, size, and nullability
```

```
Field1(0) = "Name" ' Column name.
```

```
Field1(1) = CInt(129) ' Column type.
```

```
Field1(2) = CInt(40) ' Column size.
```

```
Field1(3) = False ' Nullable?
```

```
Field2(0) = "Age"
```

```
Field2 (1) = CInt(3)
```

```
Field2 (2) = CInt(-1)
```

```
Field2 (3) = True
```

```
Field3 (0) = "DateOfBirth"
```

```
Field3 (1) = CInt(7)
```

```
Field3 (2) = CInt(-1)
```

```
Field3 (3) = True
```

```
' put all fields into an array of arrays
```

```
Record(0) = Field1
Record(1) = Field2
Record(2) = Field3

Dim NewRs

Set NewRS = DC1.CreateRecordset(Record)

Dim fields(2)

fields(0) = Field1(0)
fields(1) = Field2(0)
fields(2) = Field3(0)

' Populate the new recordset with data values.

Dim fieldVals(2)

' Use AddNew to add the records.

fieldVals(0) = "Joe"
fieldVals(1) = 5
fieldVals(2) = CDate(#1/5/96#)
NewRS.AddNew fields, fieldVals

fieldVals(0) = "Mary"
fieldVals(1) = 6
fieldVals(2) = CDate(#6/5/94#)
NewRS.AddNew fields, fieldVals

fieldVals(0) = "Alex"
fieldVals(1) = 13
fieldVals(2) = CDate(#1/6/88#)
NewRS.AddNew fields, fieldVals
```

```
fieldVals(0) = "Susan"
fieldVals(1) = 13
fieldVals(2) = CDate(#8/6/87#)
NewRS.AddNew fields, fieldVals
NewRS.MoveFirst
' Set the newly created and populated Recordset to
' the SourceRecordset property of the
' RDS.DataControl to bind to visual controls
Set DC1.SourceRecordset = NewRS
End Sub
</script>
<table datasrc="#DC1" align="center">
<thead>
<tr id="ColHeaders" class="thead2">
<th>Name</th>
<th>Age</th>
<th>D.O.B.</th>
</tr>
</thead>
<tbody class="tbody">
<tr>
<td><input datafld="Name" size=15 id=text1 name=text1> </td>
<td><input datafld="Age" size=25 id=text2 name=text2> </td>
<td><input datafld="DateOfBirth" size=25 id=text3 name=text3> </td>
```

```
</tr>
</tbody>
</table>
<input type = "button" onclick = "GetRS()" value="Go!">
</body>
</html>
<!-- EndCreateRecordsetVBS -->
```

See Also

[CreateRecordset Method \(RDS\)](#) | [Recordset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

DataControl Object Example (VBScript)

The following code shows how to set the [RDS.DataControl](#) parameters at design time and [bind](#) them to a data-aware control. Cut and paste this code between the <Body> and </Body> tags in a normal HTML document and name it **DataControlDesignVBS.asp**. ASP script will identify your server.

```
<!-- BeginDataControlDesignVBS -->

<%@ Language=VBScript %>

<HTML>

<HEAD>

<META name="VI60_DefaultClientScript" content=VBScript>

<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">

<title>RDS DataControl</title>

<% ' local style sheet used for display%>

<STYLE>

<!--

BODY {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;

COLOR:black;

}

.thead {

background-color: #008080;
```

```
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</STYLE>
</HEAD>
<BODY>
<H2>RDS API Code Examples</H2>
<HR>
<H3>Remote Data Service</H3>
<TABLE DATASRC=#RDS>
```

```

<TBODY>

<TR>

<TD><SPAN DATAFLD="FirstName"></SPAN></TD>

<TD><SPAN DATAFLD="LastName"></SPAN></TD>

</TR>

</TBODY>

</TABLE>

<!-- Remote Data Service with Parameters set at Design Time -->

<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
ID=RDS>

<PARAM NAME="SQL" VALUE="Select * from Employees for browse">

<PARAM NAME="SERVER"
VALUE="http://<%=Request.ServerVariables("SERVER_NAME")%>">

<PARAM NAME="CONNECT" VALUE="Provider='sqloledb';Integrated
Security='SSPI';Initial Catalog='Northwind'">

</OBJECT>

</BODY>

</HTML>

<!-- EndDataControlDesignVBS -->

```

The following example shows how to set the necessary parameters of **RDS.DataControl** at run time. To test this example, cut and paste this code between the <Body> and </Body> tags in a normal HTML document and name it **DataControlRuntimeVBS.asp**. ASP script will identify your server.

```

<!-- BeginDataControlRuntimeVBS -->

<%@ Language=VBScript %>

<html>

```

```
<head>
<meta name="VI60_DefaultClientScript" content=VBScript>
<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">
<title>Data Control Object Example (VBScript)</title>
<%' local style sheet used for display%>
<style>
<!--
body {
font-family: 'Verdana','Arial','Helvetica',sans-serif;
BACKGROUND-COLOR:white;
COLOR:black;
}
.thead {
background-color: #008080;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
```

```
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</style>
</head>
<body>
<h1>Data Control Object Example (VBScript)</h1>
<H2>RDS API Code Examples</H2>
<HR>
<H3>Remote Data Service Run Time</H3>
<TABLE DATASRC=#RDS>
<TBODY>
<TR>
<TD><SPAN DATAFLD="au_lname"></SPAN></TD>
<TD><SPAN DATAFLD="au_fname"></SPAN></TD>
</TR>
</TBODY>
</TABLE>
<% ' RDS.DataControl with no parameters set at design time %>
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID=RDS
```

```
HEIGHT=1 WIDTH=1></OBJECT>
```

```
<FORM name="frmInput">
```

```
<HR>
```

```
<Input Size="70" Name="txtServer"  
Value="http://<%=Request.ServerVariables("SERVER_NAME")%>"><BR>
```

```
<Input Size="100" Name="txtConnect" Value="Provider='sqloledb';Data  
Source=<%=Request.ServerVariables("SERVER_NAME")%>;Initial  
Catalog='Pubs';Integrated Security='SSPI';">
```

```
<BR>
```

```
<Input Size="70" Name="txtSQL" Value="Select * from Authors">
```

```
<HR>
```

```
<INPUT TYPE="BUTTON" NAME="Run" VALUE="Run"><BR>
```

```
<H4>Show grid with these values or change them to see data from  
another ODBC data source on your server</H4>
```

```
</FORM>
```

```
<Script Language="VBScript">
```

```
' Set parameters of RDS.DataControl at Run Time
```

```
Sub Run_OnClick
```

```
RDS.Server = document.frmInput.txtServer.Value
```

```
RDS.Connect = document.frmInput.txtConnect.Value
```

```
RDS.SQL = document.frmInput.txtSQL.Value
```

```
RDS.Refresh
```

```
End Sub
```

```
</Script>
```

```
</body>
```

```
</html>
```

<!-- EndDataControlRuntimeVBS -->

See Also

[DataControl Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

DataSpace Object and CreateObject Method Example (VBScript)

The following example shows how to use the [CreateObject](#) method of the [RDS.DataSpace](#) with the default business object, [RDS.Server.DataFactory](#). To test this example, cut and paste this code between the <Body> and </Body> tags in a normal HTML document and name it **DataSpaceVBS.asp**. ASP script will identify your server.

```
<!-- BeginDataSpaceVBS -->

<html>

<head>

<!--use the following META tag instead of advbs.inc-->

<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-
00AA006D2EA4" -->

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>DataSpace Object and CreateObject Method Example (VBScript)
</title>

<style>

<!--
body {
font-family: 'Verdana','Arial','Helvetica',sans-serif;
BACKGROUND-COLOR:white;
COLOR:black;
}
```

```
.thead {
background-color: #008080;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}

.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}

.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}

-->
</style>
</head>
<body>
<h1>DataSpace Object and CreateObject Method Example (VBScript)
</h1>
```

<H2>RDS API Code Examples</H2>

<HR>

<H3>Using Query Method of RDSServer.DataFactory</H3>

<!-- RDS.DataSpace ID rdsDS-->

<OBJECT ID="rdsDS" WIDTH=1 HEIGHT=1

CLASSID="CLSID:BD96C556-65A3-11D0-983A-00C04FC29E36">

</OBJECT>

<!-- RDS.DataControl with parameters set at run time -->

<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"

ID=RDS WIDTH=1 HEIGHT=1>

</OBJECT>

<TABLE DATASRC=#RDS>

<TBODY>

<TR>

<TD></TD>

<TD></TD>

</TR>

</TBODY>

</TABLE>

<HR>

<INPUT TYPE=BUTTON NAME="Run" VALUE="Run">

<H4>Click Run -

The *CreateObject* Method of the RDS.DataSpace Object Creates an instance of the RDSServer.DataFactory.

The *Query* Method of the RDS.Server.DataFactory is used to bring back a Recordset.</H4>

```
<Script Language="VBScript">
```

```
Dim rdsDF
```

```
Dim strServer
```

```
Dim strCnxn
```

```
Dim strSQL
```

```
strServer = "http://<%=Request.ServerVariables("SERVER_NAME")%>"
```

```
strCnxn = "Provider='sqloledb';Data Source=" & _
```

```
"<%=Request.ServerVariables("SERVER_NAME")%>" & ";" & _
```

```
"Integrated Security='SSPI';Initial Catalog='Northwind';"
```

```
strSQL = "Select FirstName, LastName from Employees"
```

```
Sub Run_OnClick()
```

```
Dim rs
```

```
' Create Data Factory
```

```
Set rdsDF = rdsDS.CreateObject("RDS.Server.DataFactory", strServer)
```

```
'Get Recordset
```

```
Set rs = rdsDF.Query(strCnxn, strSQL)
```

```
' Use RDS.DataControl to bind Recordset to data-aware Table above
```

```
RDS.SourceRecordset = rs
```

```
End Sub
```

```
</Script>
```

```
</body>
```

```
</html>
```

```
<!-- EndDataSpaceVBS -->
```

The following example shows how to use the **CreateObject** method to create an instance of a custom business object, VbBusObj.VbBusObjCls. It also uses the Active Server Pages scripting to identify the [Web server](#) name.

To see the complete example, open the sample applications selector. In the **Client Tier** column, select **VBScript in Internet Explorer**. In the **Middle Tier** column, select **Custom Visual Basic Business Object**.

```
Sub Window_OnLoad()  
    strServer = "http://<%=Request.ServerVariables("SERVER_NAME")%>"  
    Set BO = ADS1.CreateObject("VbBusObj.VbBusObjCls", strServer)  
    txtConnect.Value = "dsn=Pubs;uid=MyUserId;pwd=MyPassword;"  
    txtGetRecordset.Value = "Select * From authors for Browse"  
End Sub
```

See Also

[CreateObject Method \(RDS\)](#) | [DataSpace Object \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

DataFactory Object, Query Method, and CreateObject Method Example (VBScript)

This example creates an [RDS.Server.DataFactory](#) object using the [CreateObject](#) method of the [RDS.DataSpace](#) object. To test this example, cut and paste this code between the <Body> and </Body> tags in a normal HTML document and name it **DataFactoryVBS.asp**. ASP script will identify your server.

```
<!-- BeginDataFactoryVBS -->

<HTML>

<HEAD>

<!--use the following META tag instead of adovbs.inc-->

<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2EA4" -->

<META name="VI60_DefaultClientScript" Content="VBScript">

<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">

<TITLE>DataFactory Object, Query Method, and
CreateObject Method Example (VBScript)</TITLE>

<style>

<!--
body {
font-family: 'Verdana','Arial','Helvetica',sans-serif;
BACKGROUND-COLOR:white;
COLOR:black;
```

```
}  
.thead {  
background-color: #008080;  
font-family: 'Verdana','Arial','Helvetica',sans-serif;  
font-size: x-small;  
color: white;  
}  
.thead2 {  
background-color: #800000;  
font-family: 'Verdana','Arial','Helvetica',sans-serif;  
font-size: x-small;  
color: white;  
}  
.tbody {  
text-align: center;  
background-color: #f7efde;  
font-family: 'Verdana','Arial','Helvetica',sans-serif;  
font-size: x-small;  
}  
-->  
</style>  
</HEAD>  
<BODY>  
<h1>DataFactory Object, Query Method, and
```

CreateObject Method Example (VBScript)</h1>

<H2>RDS API Code Examples</H2>

<HR>

<H3>Using Query Method of RDS.Server.DataFactory</H3>

<!-- RDS.DataSpace ID RDS1-->

<OBJECT ID="RDS1" WIDTH=1 HEIGHT=1

CLASSID="CLSID:BD96C556-65A3-11D0-983A-00C04FC29E36">

</OBJECT>

<!-- RDS.DataControl with parameters

set at run time -->

<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"

ID=RDS WIDTH=1 HEIGHT=1>

</OBJECT>

<TABLE DATASRC=#RDS>

<TBODY>

<TR>

<TD></TD>

<TD></TD>

</TR>

</TBODY>

</TABLE>

<HR>

<INPUT TYPE=BUTTON NAME="Run" VALUE="Run">

<H4>Click Run -

The <i>CreateObject</i> Method of the RDS.DataSpace Object Creates an instance of the RDSServer.DataFactory;

The <i>Query</i> Method of the RDSServer.DataFactory is used to bring back a Recordset. </H4>

```
<Script Language="VBScript">
```

```
Dim rdsDF
```

```
Dim strServer
```

```
Dim strCnxn
```

```
Dim strSQL
```

```
strServer = "http://<%=Request.ServerVariables("SERVER_NAME")%>"
```

```
strCnxn = "Provider='sqloledb';Integrated Security='SSPI';Initial Catalog='Northwind';"
```

```
strSQL = "Select FirstName, LastName from Employees"
```

```
Sub Run_OnClick()
```

```
' Create RDSServer.DataFactory Object
```

```
Dim rs
```

```
' Get Recordset
```

```
Set DF = RDS1.CreateObject("RDSServer.DataFactory", strServer)
```

```
Set rs = DF.Query(strCnxn, strSQL)
```

```
' Set parameters of RDS.DataControl at Run Time
```

```
RDS.Server = strServer
```

```
RDS.SQL = strSQL
```

```
RDS.Connect = strCnxn
```

```
RDS.Refresh
```

```
End Sub
```

```
</Script>
```

```
</BODY>
```

```
</HTML>
```

```
<!-- EndDataFactoryVBS -->
```

See Also

[CreateObject Method \(RDS\)](#) | [DataFactory Object \(RDS Server\)](#) | [DataSpace Object \(RDS\)](#) | [Query Method \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

ExecuteOptions and FetchOptions Properties Example (VBScript)

The following code shows how to set the [ExecuteOptions](#) and [FetchOptions](#) properties at design time. If left unset, **ExecuteOptions** defaults to **adcExecSync**. This setting indicates that when the **RDS.Refresh** method is called, it will be executed on the current calling thread—that is, [synchronously](#). Cut and paste the following code to Notepad or another text editor and save it as **ExecuteOptionsDesignVBS.asp**.

```
<!-- BeginExecuteOptionsDesignVBS -->

<%@ Language=VBScript %>

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>Design-time ExecuteOptions and FetchOptions Properties
Example</title>

<style>

<!--

body {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;

COLOR:black;

}

.thead {
```

```
background-color: #008080;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</style>
</head>
<body>
<h2>Design-time <br> ExecuteOptions and FetchOptions Properties
Example</h2>
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID=RDS
height=1 width=1>
```

```
<PARAM NAME="SQL" VALUE="SELECT FirstName, LastName FROM Employees  
ORDER BY LastName">
```

```
<PARAM NAME="Connect" VALUE="Provider='sqloledb';Data Source=  
<%=Request.ServerVariables("SERVER_NAME")%>;Integrated  
Security='SSPI';Initial Catalog='Northwind'">
```

```
<PARAM NAME="Server"  
VALUE="http://<%=Request.ServerVariables("SERVER_NAME")%>">
```

```
<PARAM NAME="ExecuteOptions" VALUE="1">
```

```
<PARAM NAME="FetchOptions" VALUE="3">
```

```
</OBJECT>
```

```
<TABLE DATASRC=#RDS>
```

```
<TBODY>
```

```
<TR class="thead2">
```

```
<TH>First Name</TH>
```

```
<TH>Last Name</TH>
```

```
</TR>
```

```
<TR class="tbody">
```

```
<TD><SPAN DATAFLD="FirstName"></SPAN></TD>
```

```
<TD><SPAN DATAFLD="LastName"></SPAN></TD>
```

```
</TR>
```

```
</TBODY>
```

```
</TABLE>
```

```
</body>
```

```
</html>
```

```
<!-- EndExecuteOptionsDesignVBS -->
```

The following example shows how to set the **ExecuteOptions** and

FetchOptions properties at run time in VBScript code. See the [Refresh](#) method for a working example of these properties. Cut and paste the following code to Notepad or another text editor and save it as **ExecuteOptionsRuntimeVBS.asp**.

```
<!-- BeginExecuteOptionsRuntimeVBS -->

<%@ Language=VBScript %>

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>Run-time ExecuteOptions and FetchOptions Properties
Example</title>

<style>

<!--

body {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;

COLOR:black;

}

.thead {

background-color: #008080;

font-family: 'Verdana','Arial','Helvetica',sans-serif;

font-size: x-small;

color: white;

}

.thead2 {
```

```
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</style>
</head>
<body>
<h2>Run-time <br> ExecuteOptions and FetchOptions Properties
Example</h2>
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID=RDS
height=1 width=1>
<PARAM NAME="SQL" VALUE="SELECT FirstName, LastName FROM Employees
ORDER BY LastName">
<PARAM NAME="Connect" VALUE="Provider='sqloledb';Data Source=
<%=Request.ServerVariables("SERVER_NAME")%>;Integrated
Security='SSPI';Initial Catalog='Northwind'">
<PARAM NAME="Server"
VALUE="http://<%=Request.ServerVariables("SERVER_NAME")%>">
</OBJECT>
```

```
<TABLE DATASRC=#RDS>
<TBODY>
<TR class="thead2">
<TH>First Name</TH>
<TH>Last Name</TH>
</TR>
<TR class="tbody">
<TD><SPAN DATAFLD="FirstName"></SPAN></TD>
<TD><SPAN DATAFLD="LastName"></SPAN></TD>
</TR>
</TBODY>
</TABLE>
<Script Language="VBScript">
Const adcExecSync = 1
Const adcFetchAsynch = 3
Sub ExecuteHow
' set RDS properties at run-time
RDS1.ExecuteOptions = adcExecSync
RDS1.FetchOptions = adcFetchAsynch
RDS.Refresh
End Sub
</Script>
</body>
</html>
```

<!-- EndExecuteOptionsRuntimeVBS -->

See Also

[ExecuteOptions Property \(RDS\)](#) | [FetchOptions Property \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

FilterColumn, FilterCriterion, FilterValue, SortColumn, and SortDirection Properties and Reset Method Example (VBScript)

The following code shows how to set the [RDS.DataControl Server](#) parameter at design_time and bind it to a data-aware HTML table using a data source. Cut and paste the following code to Notepad or another text editor and save it as **FilterColumnVBS.asp**.

```
<!-- BeginFilterColumnVBS -->

<%@ Language=VBScript %>

<HTML>

<HEAD>

<META name="VI60_DefaultClientScript" Content="VBScript">

<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">

<TITLE>FilterColumn, FilterCriterion, FilterValue, SortColumn, and
SortDirection
Properties and Reset Method Example (VBScript)</TITLE>

</HEAD>

<BODY>

<h1>FilterColumn, FilterCriterion, FilterValue, SortColumn, and
SortDirection
Properties and Reset Method Example (VBScript)</h1>

<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID=RDS
HEIGHT=1 WIDTH=1>
```

```
<PARAM NAME="SQL" VALUE="Select FirstName, LastName, Title, ReportsTo, Extension from Employees">
```

```
<PARAM NAME="Connect" VALUE="Provider='sqloledb';Data Source=<%=Request.ServerVariables("SERVER_NAME")%>;Integrated Security='SSPI';Initial Catalog='Northwind'">
```

```
<PARAM NAME="Server" VALUE="http://<%=Request.ServerVariables("SERVER_NAME")%>">
```

```
</OBJECT>
```

```
Sort Column: <SELECT NAME="cboSortColumn">
```

```
<OPTION VALUE=""></OPTION>
```

```
<OPTION VALUE=ID>ID</OPTION>
```

```
<OPTION VALUE=FirstName>FirstName</OPTION>
```

```
<OPTION VALUE=LastName>LastName</OPTION>
```

```
<OPTION VALUE=Title>Title</OPTION>
```

```
<OPTION VALUE=Title>ReportsTo</OPTION>
```

```
<OPTION VALUE=Phone>Extension</OPTION>
```

```
</SELECT>
```

```
<br>
```

```
Sort Direction: <SELECT NAME="cboSortDir">
```

```
<OPTION VALUE=""></OPTION>
```

```
<OPTION VALUE=TRUE>Ascending</OPTION>
```

```
<OPTION VALUE=FALSE>Descending</OPTION>
```

```
</SELECT>
```

```
<HR WIDTH="25%">
```

```
Filter Column: <SELECT NAME="cboFilterColumn">
```

```
<OPTION VALUE=""></OPTION>
```

<OPTION VALUE=FirstName>FirstName</OPTION>

<OPTION VALUE=LastName>LastName</OPTION>

<OPTION VALUE=Title>Title</OPTION>

<OPTION VALUE=Room>ReportsTo</OPTION>

<OPTION VALUE=Phone>Extension</OPTION>

</SELECT>

Filter Criterion: <SELECT NAME="cboCriterion">

<OPTION VALUE=""></OPTION>

<OPTION VALUE="">=</OPTION>

<OPTION VALUE=">">></OPTION>

<OPTION VALUE="<"><</OPTION>

<OPTION VALUE=">=">>=</OPTION>

<OPTION VALUE="<="><=</OPTION>

<OPTION VALUE="<>"><></OPTION>

</SELECT>

Filter Value: <INPUT NAME="txtFilterValue">

<HR WIDTH="25%">

<INPUT TYPE=BUTTON NAME=Clear VALUE="CLEAR ALL">

<INPUT TYPE=BUTTON NAME=SortFilter VALUE="APPLY">

<HR>

<TABLE DATASRC=#RDS ID="DataTable">

<THEAD>

```
<TR>
<TH>FirstName</TH>
<TH>LastName</TH>
<TH>Title</TH>
<TH>Reports To</TH>
<TH>Extension</TH>
</TR>
</THEAD>
<TBODY>
<TR>
<TD><SPAN DATAFLD="FirstName"></SPAN></TD>
<TD><SPAN DATAFLD="LastName"></SPAN></TD>
<TD><SPAN DATAFLD="Title"></SPAN></TD>
<TD><SPAN DATAFLD="ReportsTo"></SPAN></TD>
<TD><SPAN DATAFLD="Extension"></SPAN></TD>
</TR>
</TBODY>
</TABLE>
```

```
<Script Language="VBScript">
```

```
<!--
```

```
Const adFilterNone = 0
```

```
Sub SortFilter_OnClick
```

```
Dim vCriterion
```

```
Dim vSortDir
```

```

Dim vSortCol

Dim vFilterCol

' The value of SortColumn will be the
' value of what the user picks in the
' cboSortColumn box.
vSortCol = cboSortColumn.options(cboSortColumn.selectedIndex).value
If(vSortCol <> "") then
RDS.SortColumn = vSortCol
End If

' The value of SortDirection will be the
' value of what the user specifies in the
' cboSortdirection box.
If (vSortCol <> "") then
vSortDir = cboSortDir.options(cboSortDir.selectedIndex).value
If (vSortDir = "") then
MsgBox "You must select a direction for the sort."
Exit Sub
Else
If vSortDir = "Ascending" Then vSortDir = "TRUE"
If vSortDir = "Descending" Then vSortDir = "FALSE"
RDS.SortDirection = vSortDir
End If
End If

' The value of FilterColumn will be the

```

```
' value of what the user specifies in the
' cboFilterColumn box.

vFilterCol =
cboFilterColumn.options(cboFilterColumn.selectedIndex).value

If(vFilterCol <> "") then

RDS.FilterColumn = vFilterCol

End If

' The value of FilterCriterion will be the
' text value of what the user specifies in the
' cboCriterion box.

vCriterion = cboCriterion.options(cboCriterion.selectedIndex).value

If (vCriterion <> "") Then

RDS.FilterCriterion = vCriterion

End If

' txtFilterValue is a rich text box
' control. The value of FilterValue will be the
' text value of what the user specifies in the
' txtFilterValue box.

If (txtFilterValue.value <> "") Then

RDS.FilterValue = txtFilterValue.value

End If

' Execute the sort and filter on a client-side
' Recordset based on the specified sort and filter
' properties. Calling Reset refreshes the result set
```

```
' that is displayed in the data-bound controls to  
' display the filtered, sorted recordset.
```

```
RDS.Reset
```

```
End Sub
```

```
Sub Clear_onClick()
```

```
'clear the HTML input controls
```

```
cboSortColumn.selectedIndex = 0
```

```
cboSortDir.selectedIndex = 0
```

```
cboFilterColumn.selectedIndex = 0
```

```
cboCriterion.selectedIndex = 0
```

```
txtFilterValue.value = ""
```

```
'clear the filter
```

```
RDS.FilterCriterion = ""
```

```
RDS.Reset(FALSE)
```

```
End Sub
```

```
-->
```

```
</Script>
```

```
</BODY>
```

```
</HTML>
```

```
<!-- EndFilterColumnVBS -->
```

See Also

[DataControl Object \(RDS\)](#) | [FilterColumn Property \(RDS\)](#) | [FilterCriterion Property \(RDS\)](#) | [FilterValue Property \(RDS\)](#) | [Reset Method \(RDS\)](#) | [SortColumn Property \(RDS\)](#) | [SortDirection Property \(RDS\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

ReadyState Property Example (VBScript)

The following example shows how to read the [ReadyState](#) property of the [RDS.DataControl](#) object at run time in VBScript code. **ReadyState** is a read-only property.

To test this example, cut and paste this code between the <Body> and </Body> tags in a normal HTML document and name it **RDSReadySt.asp**. Use **Find** to locate the file Adovbs.inc and place it in the directory you plan to use. ASP script will identify your server.

```
<!-- BeginReadyStateVBS -->

<%@ Language=VBScript %>

<% 'use the following META tag instead of adovbs.inc%>

<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-
00AA006D2EA4" -->

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>RDS.DataControl ReadyState Property</title>

<style>

<!--

body {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;
```

```
COLOR:black;
}
.thead {
background-color: #008080;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</style>
</head>
<body>
```

<H1>RDS.DataControl ReadyState Property</H1>

<H2>RDS API Code Examples </H2>

<HR>

<!-- RDS.DataControl with parameters set at design time -->

<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
ID=RDS>

<PARAM NAME="SQL" VALUE="Select * from Orders">

<PARAM NAME="SERVER"
VALUE="http://<%=Request.ServerVariables("SERVER_NAME")%>">

<PARAM NAME="CONNECT" VALUE="Provider='sqloledb';Integrated
Security='SSPI';Initial Catalog='Northwind'">

<PARAM NAME="ExecuteOptions" VALUE="2">

<PARAM NAME="FetchOptions" VALUE="3">

</OBJECT>

<TABLE DATASRC=#RDS>

<TBODY>

<TR>

<TD></TD>

</TR>

</TBODY>

</TABLE>

<Script Language="VBScript">

Sub Window_OnLoad

Select Case RDS.ReadyState

case 2 'adcReadyStateLoaded

```
MsgBox "Executing Query"
case 3 'adcReadyStateInteractive
MsgBox "Fetching records in background"
case 4 'adcReadyStateComplete
MsgBox "All records fetched"
End Select
End Sub
</Script>
</body>
</html>
<!-- EndReadyStateVBS -->
```

See Also

[DataControl Object \(RDS\)](#) | [ReadyState Property \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Recordset and SourceRecordset Properties Example (VBScript)

The following example shows how to set the necessary parameters of the [RDS.Server.DataFactory](#) default [business object](#) at run time.

To test this example, cut and paste this code between the <Body> and </Body> tags in a normal HTML document and name it **RecordsetVBS.asp**. ASP script will identify your server.

```
<!-- BeginRecordSetVBS -->

<%@ Language=VBScript %>

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>Recordset and SourceRecordset Properties Example (VBScript)
</title>

<style>

<!--
body {
font-family: 'Verdana','Arial','Helvetica',sans-serif;
BACKGROUND-COLOR:white;
COLOR:black;
}
.thead {
```

```
background-color: #008080;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</style>
</head>
<body>
<h1>Recordset and SourceRecordset Properties Example (VBScript)
</h1>
<Center>
```

<H2>RDS API Code Examples</H2>

<HR>

<H3>Using SourceRecordset and Recordset with
RDS.Server.DataFactory</H3>

<!-- RDS.DataSpace ID RDS1 -->

<OBJECT ID="RDS1" WIDTH=1 HEIGHT=1

CLASSID="CLSID:BD96C556-65A3-11D0-983A-00C04FC29E36">

</OBJECT>

<!-- RDS.DataControl with parameters set at Run Time -->

<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"

ID=RDC WIDTH=1 HEIGHT=1>

</OBJECT>

<TABLE DATASRC=#RDC>

<TR>

<TD> <INPUT DATAFLD="FirstName" SIZE=15> </TD>

<TD> <INPUT DATAFLD="LastName" SIZE=15></TD>

</TR>

</TABLE>

<HR>

<Input Size=70 Name="txtServer"
Value="http://<%=Request.ServerVariables("SERVER_NAME")%>">

<Input Size=70 Name="txtConnect"
Value="Provider='sqloledb';Integrated Security='SSPI';Initial
Catalog='Northwind'">

<Input Size=70 Name="txtSQL" Value="SELECT FirstName, LastName FROM
Employees">

```
<HR>
<INPUT TYPE=BUTTON NAME="Run" VALUE="Run"><BR>
</Center>
<Script Language="VBScript">
Dim rdsDF
Dim strServer
strServer = "http://<%=Request.ServerVariables("SERVER_NAME")%>"
Sub Run_OnClick()
Dim rs
' Create RDS.Server.DataFactory Object
Set rdsDF = RDS1.CreateObject("RDS.Server.DataFactory", strServer)
' Get Recordset
Set rs = rdsDF.Query(txtConnect.Value,txtSQL.Value)
' Set parameters of RDS.DataControl at run time
RDC.Server = txtServer.Value
RDC.SQL = txtSQL.Value
RDC.Connect = txtConnect.Value
RDC.Refresh
End Sub
</Script>
</body>
</html>
<!-- EndRecordsetVBS -->
```

See Also

[DataFactory Object \(RDS Server\)](#) | [Recordset, SourceRecordset Properties \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Refresh Method Example (VBScript)

The following example shows how to set the necessary parameters of [RDS.DataControl](#) at run time. The manner in which a [Recordset](#) is retrieved using the [Refresh](#) method is determined by the settings of the [ExecuteOptions](#) and [FetchOptions](#) properties. To test this example, cut and paste the following code into a normal ASP document and name it **RefreshVBS.asp**. Use **Find** to locate the file Adovbs.inc and place it in the directory you plan to use. ASP script will identify your server.

```
<!-- BeginRefreshVBS -->

<%@ Language=VBScript %>

<!--use the following META tag instead of adovbs.inc-->

<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-00AA006D2EA4" -->

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>Refresh Method Example (VBScript)</title>

<style>

<!--

body {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;

COLOR:black;

}
```

```
.thead {
background-color: #008080;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}

.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}

.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}

-->

</style>

</head>

<body>

<h1>Refresh Method Example (VBScript)</h1>

<H2>RDS API Code Examples </H2>
```

<HR>

<TABLE DATASRC=#RDC>

<TR>

<TD> <INPUT DATAFLD="FirstName" SIZE=15> </TD>

<TD> <INPUT DATAFLD="LastName" SIZE=15> </TD>

<TD> <INPUT DATAFLD="Title" SIZE=15> </TD>

<TD> <INPUT DATAFLD="HireDate" SIZE=15> </TD>

</TR>

</TABLE>

<!-- RDS.DataControl with no parameters set at design time -->

<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"

ID=RDC HEIGHT=1 WIDTH=1>

</OBJECT>

<HR>

Server: <Input Size=70 Name="txtServer"
Value="http://<%=Request.ServerVariables("SERVER_NAME")%>">

Connect: <Input Size=70 Name="txtConnect"
Value="Provider='sqloledb';Integrated Security='SSPI';Initial
Catalog='Northwind'">

SQL: <Input Size=70 Name="txtSQL" Value="Select * from Employees">

<HR>

<TABLE BORDER=1 WIDTH="60%">

<TR>

<TD COLSPAN=3 BGCOLOR=silver>

Choose if you want the Recordset brought back Synchronously on the

current calling thread or Asynchronously on another thread.

</TD>

</TR>

<TR>

<TD>Synchronously:

<Input Type="Radio" Name="optExecuteOptions" Checked
OnClick="SetEx0('adcExecSync')">

</TD>

<TD>Asynchronously:

<Input Type="Radio" Name="optExecuteOptions"
OnClick="SetEx0('adcExecAsync')">

</TD>

<TD> </TD>

</TR>

<TR>

<TD COLSPAN=3 BGCOLOR=silver>

Fetch Up Front, Background Fetch with Blocking or Background Fetch
without Blocking

</TD>

<TR>

<TD>Up Front:

<Input Type="Radio" Name="optFetchOptions"
OnClick="SetF0('adcFetchUpFront')">

</TD>

<TD>Background w/ Blocking:


```
<Input Type="Radio" Name="optFetchOptions" Checked  
OnClick="SetF0('adcFetchBackground')">
```

```
</TD>
```

```
<TD>Background w/o Blocking:<BR>
```

```
<Input Type="Radio" Name="optFetchOptions"  
OnClick="SetF0('adcFetchAsync')">
```

```
</TD>
```

```
</TR>
```

```
</TABLE>
```

```
<INPUT TYPE=BUTTON NAME="Run" VALUE="Run"><BR>
```

```
<Script Language="VBScript">
```

```
<!--
```

```
Dim E0 'ExecuteOptions
```

```
Dim F0 'FetchOptions
```

```
E0 = "adcExecSync" 'Default value
```

```
F0 = "adcFetchBackground" 'Default value
```

```
Sub SetEx0(NewE0)
```

```
E0 = NewE0
```

```
End Sub
```

```
Sub SetF0(NewF0)
```

```
F0 = NewF0
```

```
End Sub
```

```
' Set parameters of RDS.DataControl at Run Time
```

```
Sub Run_OnClick
```

```
RDC.Server = txtServer.Value
```

```
RDC.SQL = txtSQL.Value

RDC.Connect = txtConnect.Value

If EO = "adcExecSync" Then 'Determine which ExecuteOption chosen

RDC.ExecuteOptions = adcExecSync

MsgBox "Recordset brought in on current calling thread
Synchronously"

Else

RDC.ExecuteOptions = adcExecAsync

MsgBox "Recordset brought in on another thread Asynchronously"

End If

If FO = "adcFetchBackground" Then 'Determine 'which FetchOption
chosen

RDC.FetchOptions = adcFetchBackground

MsgBox "Control goes back to user after first batch of records
returned"

ElseIf FO = " adcFetchUpFront" Then

RDC.FetchOptions = adcFetchUpFront

MsgBox "All records returned before control goes back to user"

Else

RDC.FetchOptions = adcFetchAsync

MsgBox "Control goes back to user immediately"

End If

RDC.Refresh

End Sub

-->
```

```
</Script>  
</body>  
</html>  
<!-- EndRefreshVBS -->
```

See Also

[DataControl Object \(RDS\)](#) | [ExecuteOptions Property \(RDS\)](#) | [FetchOptions Property \(RDS\)](#) | [Recordset Object](#) | [Refresh Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Server Property Example (VBScript)

The following code shows how to set the [RDS.DataControl](#) parameter at design time and bind it to a data-aware control using the SQLOLEDB provider. Cut and paste this code into a normal ASP document and name it **ServerDesignVBS.asp**. ASP script will identify your server.

```
<!-- BeginServerDesignVBS -->

<%@ Language=VBScript %>

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>Server Property Example (VBScript)</title>

<style>

<!--

body {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;

COLOR:black;

}

.thead {

background-color: #008080;

font-family: 'Verdana','Arial','Helvetica',sans-serif;

font-size: x-small;
```

```
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana', 'Arial', 'Helvetica', sans-serif;
font-size: x-small;
}
-->
</style>
</head>
<body>
<h1>Server Property Example (VBScript)</h1>
<TABLE DATASRC=#RDS>
<TR>
<TD> <SPAN DATAFLD="FirstName"></SPAN> </TD>
<TD> <SPAN DATAFLD="LastName"></SPAN> </TD>
<TD> <SPAN DATAFLD="Title"></SPAN> </TD>
```

```

<TD> <SPAN DATAFLD="Type"></SPAN> </TD>
<TD> <SPAN DATAFLD="Email"></SPAN> </TD>
</TR>
</TABLE>
<!-- Remote Data Service with Parameters set at Design Time -->
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
ID=RDS HEIGHT=1 WIDTH=1>
<PARAM NAME="SQL" VALUE="Select * from Employees">
<PARAM NAME="SERVER"
VALUE="http://<%=Request.ServerVariables("SERVER_NAME")%>">
<PARAM NAME="CONNECT" VALUE="Provider='sqloledb';Integrated
Security='SSPI';Initial Catalog='Northwind'">
</OBJECT>
</body>
</html>
<!-- EndServerDesignVBS -->

```

The following example shows how to set the necessary parameters of **RDS.DataControl** at run time. To test this example, cut and paste this code into a normal ASP document and name it **ServerRuntimeVBS.asp**. ASP script will identify your server.

```

<!-- BeginServerRuntimeVBS -->
<%@ Language=VBScript %>
<html>
<head>
<meta name="VI60_DefaultClientScript" content=VBScript>
<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

```

```
<title>Server Property Example (VBScript)</title>
<style>
<!--
body {
font-family: 'Verdana','Arial','Helvetica',sans-serif;
BACKGROUND-COLOR:white;
COLOR:black;
}
.thead {
background-color: #008080;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
```

```
font-size: x-small;
}
-->
</style>
</head>
<body>
<h1>Server Property Example (VBScript)</h1>
<H2>RDS API Code Examples</H2>
<H3>Remote Data Service Server Property Set at Run Time</H3>
<!-- RDS.DataControl with no parameters set at design time -->
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
ID=RDC HEIGHT=1 WIDTH=1>
</OBJECT>
<TABLE DATASRC=#RDC>
<TR>
<TD> <SPAN DATAFLD="FirstName"></SPAN> </TD>
<TD> <SPAN DATAFLD="LastName"></SPAN> </TD>
<TD> <SPAN DATAFLD="Title"></SPAN> </TD>
<TD> <SPAN DATAFLD="Type"></SPAN> </TD>
<TD> <SPAN DATAFLD="Email"></SPAN> </TD>
</TR>
</TABLE>
<HR>
<Input Size=70 Name="txtServer" Value="HTTP://<%=
```

```
Request.ServerVariables("SERVER_NAME")%>">

<BR>

<Input Size=70 Name="txtConnect"
Value="Provider='sqloledb';Integrated Security='SSPI';Initial
Catalog='Northwind'">

<BR>

<Input Size=70 Name="txtSQL" Value="Select * from Employees">

<HR>

<INPUT TYPE=BUTTON NAME="Run" VALUE="Run"><BR>

<Script Language="VBScript">

<!--

' Set parameters of RDS.DataControl at Run Time

Sub Run_OnClick

RDC.Server = txtServer.Value

RDC.SQL = txtSQL.Value

RDC.Connect = txtConnect.Value

RDC.Refresh

End Sub

-->

</Script>

</body>

</html>

<!-- EndServerRuntimeVBS -->
```

[DataControl Object \(RDS\)](#) | [Server Property \(RDS\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

SQL Property Example (VBScript)

The following code shows how to set the [RDS.DataControl](#) SQL parameter at design time and bind it to a data-aware control using the database called *Pubs*, which ships with Microsoft® SQL Server™. To test the example, copy the following code into a normal ASP document named **SQLDesignVBS.asp** on your Web server.

```
<!-- BeginSQLDesignVBS -->

<%@ Language=VBScript %>

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>SQL Property Example (VBScript)</title>

<style>

<!--

body {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;

COLOR:black;

}

.thead {

background-color: #008080;

font-family: 'Verdana','Arial','Helvetica',sans-serif;
```

```
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}
-->
</style>
</head>
<body>
<h1>SQL Property Example (VBScript)</h1>
<!-- RDS.DataControl -->
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID=RDC
HEIGHT=1 WIDTH=1>
<PARAM NAME="SQL" VALUE="Select FirstName, LastName from
Employees">
```

```

<PARAM NAME="SERVER"
VALUE="http://<%=Request.ServerVariables("SERVER_NAME")%>">

<PARAM NAME="CONNECT" VALUE="Provider='sqloledb';Initial
Catalog='Northwind';Integrated Security='SSPI';">

</OBJECT>

<!-- Data Table -->

<TABLE DATASRC=#RDC BORDER=1>

<TR>

<TD> <SPAN DATAFLD="FirstName"></SPAN> </TD>

<TD> <SPAN DATAFLD="LastName"></SPAN> </TD>

</TR>

</TABLE>

</body>

</html>

<!-- EndSQLDesignVBS -->

```

The following example shows how to set the necessary parameters of **RDS.DataControl** at run time. To test this example, cut and paste the following code into a normal ASP document and name it **SQLRuntimeVBS.asp**. ASP script will identify your server.

```

<!-- BeginServerRuntimeVBS -->

<%@ Language=VBScript %>

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>Server Property Example (VBScript)</title>

```

```
<style>
<!--
body {
font-family: 'Verdana','Arial','Helvetica',sans-serif;
BACKGROUND-COLOR:white;
COLOR:black;
}
.thead {
background-color: #008080;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}
.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
```

```
}
-->
</style>
</head>
<body>
<h1>Server Property Example (VBScript)</h1>
<H2>RDS API Code Examples</H2>
<H3>Remote Data Service Server Property Set at Run Time</H3>
<!-- RDS.DataControl with no parameters set at design time -->
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
ID=RDC HEIGHT=1 WIDTH=1>
</OBJECT>
<TABLE DATASRC=#RDC>
<TR>
<TD> <SPAN DATAFLD="FirstName"></SPAN> </TD>
<TD> <SPAN DATAFLD="LastName"></SPAN> </TD>
<TD> <SPAN DATAFLD="Title"></SPAN> </TD>
<TD> <SPAN DATAFLD="Type"></SPAN> </TD>
<TD> <SPAN DATAFLD="Email"></SPAN> </TD>
</TR>
</TABLE>
<HR>
<Input Size=70 Name="txtServer" Value="HTTP://<%=
Request.ServerVariables("SERVER_NAME")%>">
```

```

<BR>

<Input Size=70 Name="txtConnect"
Value="Provider='sqloledb';Integrated Security='SSPI';Initial
Catalog='Northwind'">

<BR>

<Input Size=70 Name="txtSQL" Value="Select * from Employees">

<HR>

<INPUT TYPE=BUTTON NAME="Run" VALUE="Run"><BR>

<Script Language="VBScript">

<!--

' Set parameters of RDS.DataControl at Run Time

Sub Run_OnClick

RDC.Server = txtServer.Value

RDC.SQL = txtSQL.Value

RDC.Connect = txtConnect.Value

RDC.Refresh

End Sub

-->

</Script>

</body>

</html>

<!-- EndServerRuntimeVBS -->

```

See Also

[DataControl Object \(RDS\)](#) | [SQL Property \(RDS\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

SubmitChanges Method Example (VBScript)

The following code fragment shows how to use the [SubmitChanges](#) method with an [RDS.DataControl](#) object.

To test this example, cut and paste this code into a normal ASP document and name it **SubmitChangesCtrlVBS.asp**. ASP script will identify your server.

```
<!-- BeginCancelUpdateVBS -->

<%@Language=VBScript%>

<%'Option Explicit%>

<% 'use the following META tag instead of adovbs.inc%>

<!--METADATA TYPE="typelib" uuid="00000205-0000-0010-8000-
00AA006D2EA4" -->

<HTML>

<HEAD>

<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">

<TITLE></TITLE>

</HEAD>

<BODY>

<CENTER>

<H1>Remote Data Service</H1>

<H2>SubmitChanges and CancelUpdate Methods</H2>

<% ' to integrate/test this code replace the Server property value
and
```

' the Data Source value in the Connect property with appropriate values%>

<HR>

<OBJECT ID=RDS classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" HEIGHT=1 WIDTH=1></OBJECT>

<SCRIPT Language="VBScript">

'set RDS properties for control just created

RDS.Server = "http://<%=Request.ServerVariables("SERVER_NAME")%>"

RDS.SQL = "Select * from Employees"

RDS.Connect = "Provider='sqloledb';Integrated Security='SSPI';Initial Catalog='Northwind';"

RDS.Refresh

</SCRIPT>

<TABLE DATASRC=#RDS>

<THEAD>

<TR ID="ColHeaders">

<TH>ID</TH>

<TH>FName</TH>

<TH>LName</TH>

<TH>Title</TH>

<TH>Hire Date</TH>

<TH>Birth Date</TH>

<TH>Extension</TH>

<TH>Home Phone</TH>

</TR>

```
</THEAD>

<TBODY>

<TR>

<TD> <INPUT DATAFLD="EmployeeID" size=4> </TD>

<TD> <INPUT DATAFLD="FirstName" size=10> </TD>

<TD> <INPUT DATAFLD="LastName" size=10> </TD>

<TD> <INPUT DATAFLD="Title" size=10> </TD>

<TD> <INPUT DATAFLD="HireDate" size=10> </TD>

<TD> <INPUT DATAFLD="BirthDate" size=10> </TD>

<TD> <INPUT DATAFLD="Extension" size=10> </TD>

<TD> <INPUT DATAFLD="HomePhone" size=8> </TD>

</TR>

</TBODY>

</TABLE>

<HR>

<INPUT TYPE=button NAME="SubmitChange" VALUE="Submit Changes">

<INPUT TYPE=button NAME="CancelChange" VALUE="Cancel Update">

<BR>

<H4>Alter a current entry on the grid. Move off that Row. <BR>

Submit the Changes to your DBMS or cancel the updates. </H4>

</CENTER>

<SCRIPT Language="VBScript">

Sub SubmitChange_OnClick

msgbox "Changes will be made"
```

```
RDS.SubmitChanges
RDS.Refresh
End Sub
Sub CancelChange_OnClick
msgbox "Changes will be cancelled"
RDS.CancelUpdate
RDS.Refresh
End Sub
-->
</SCRIPT>
</BODY>
</HTML>
<!-- EndCancelUpdateVBS -->
```

See Also

[DataControl Object \(RDS\)](#) | [SubmitChanges Method \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

URL Property Example (VBScript)

The following code demonstrates how to set the **URL** property on the client side to specify an .asp file that in turn handles the submission of changes to the data source.

```
<!-- BeginURLClientVBS -->

<%@ Language=VBScript %>

<html>

<head>

<meta name="VI60_DefaultClientScript" content=VBScript>

<meta name="GENERATOR" content="Microsoft Visual Studio 6.0">

<title>URL Property Example (VBScript)</title>

<style>

<!--

body {

font-family: 'Verdana','Arial','Helvetica',sans-serif;

BACKGROUND-COLOR:white;

COLOR:black;

}

.thead {

background-color: #008080;

font-family: 'Verdana','Arial','Helvetica',sans-serif;

font-size: x-small;

color: white;
```

```
}

.thead2 {
background-color: #800000;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
color: white;
}

.tbody {
text-align: center;
background-color: #f7efde;
font-family: 'Verdana','Arial','Helvetica',sans-serif;
font-size: x-small;
}

-->

</style>

</head>

<body onload=Getdata(>

<h1>URL Property Example (VBScript)</h1>

<OBJECT classid=clsid:BD96C556-65A3-11D0-983A-00C04FC29E33 height=1
id=ADC width=1>

</OBJECT>

<table datasrc="#ADC" align="center">

<thead>

<tr id="ColHeaders" class="thead2">
```

```
<th>FirstName</th>
<th>LastName</th>
<th>Extension</th>
</tr>
</thead>
<tbody class="tbody">
<tr>
<td><input datafld="FirstName" size=15> </td>
<td><input datafld="LastName" size=25> </td>
<td><input datafld="Extension" size=15> </td>
</tr>
</tbody>
</table>
<script Language="VBScript">
Sub Getdata()
msgbox "getdata"
ADC.URL =
"http://<%=Request.ServerVariables("SERVER_NAME")%>/URLServerVBS.asp
ADC.Refresh
End Sub
</script>
</body>
</html>
<!-- EndURLClientVBS -->
```

The server-side code that exists in **URLServerVBS.asp** submits the updated **Recordset** to the data source.

```
<!-- BeginURLServerVBS -->

<%@ Language=VBScript %>

<%

' XML output req's

Response.ContentType = "text/xml"

const adPersistXML = 1

' recordset vars

Dim strSQL, rsEmployees

Dim strCnxn, Cnxn

strCnxn = "Provider='sqloledb';Data Source=" & _
Request.ServerVariables("SERVER_NAME") & ";" & _
"Integrated Security='SSPI';Initial Catalog='Northwind';"

Set Cnxn = Server.CreateObject("ADODB.Connection")

Set rsEmployees = Server.CreateObject("ADODB.Recordset")

strSQL = "SELECT FirstName, LastName, Extension FROM Employees"

Cnxn.Open strCnxn

rsEmployees.Open strSQL, Cnxn

' output as XML

rsEmployees.Save Response, adPersistXML

' Clean up

rsEmployees.Close

Cnxn.Close
```

```
Set rsEmployees = Nothing
```

```
Set Cnxn = Nothing
```

```
%>
```

```
<!-- EndURLServerVBS -->
```

See Also

[URL Property \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

RDS Code Examples in Microsoft Visual C++

Use the following code examples to learn how to use RDS properties when writing in Microsoft Visual C++.

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

- [Handler Property Example](#)
- [InternetTimeout Property Example](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Handler Property Example (VC++)

This example demonstrates the [RDS DataControl](#) object [Handler](#) property. (See [DataFactory Customization](#) for more details.)

Assume the following sections in the parameter file, Msdfmap.ini, located on the server:

```
[connect AuthorDataBase]
Access=ReadWrite
Connect="DSN=Pubs"
[sql AuthorById]
SQL="SELECT * FROM Authors WHERE au_id = ?"
```

Your code looks like the following. The command assigned to the [SQL](#) property will match the **AuthorById** identifier and will retrieve a row for author Michael O'Leary. Although the [Connect](#) property in your code specifies the Northwind data source, that data source will be overwritten by the Msdfmap.ini *connect* section. The **DataControl** object [Recordset](#) property is assigned to a disconnected [Recordset](#) object purely as a coding convenience.

```
// BeginHandlerCpp

#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
no_namespace rename("EOF", "EndOfFile")

#import "C:\Program Files\Common Files\System\MSADC\msadco.dll"

#include <ole2.h>

#include <stdio.h>

#include <conio.h>

// Function declarations

inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};

void HandlerX(void);

void PrintProviderError(_ConnectionPtr pConnection);
```



```

{
HRESULT hr = S_OK;

// Define ADO object pointers.

// Initialize pointers on define.

// These are in the ADODB:: namespace.

_RecordsetPtr pRst = NULL;

//Define RDS object pointers.

RDS::IBindMgrPtr dc;

try
{
TESTHR(hr = dc.CreateInstance(__uuidof(RDS::DataControl)));

dc->Handler = "MSDFMAP.Handler";

dc->Server = "http://MyServer";

dc->Connect = "Data Source=AuthorDatabase";

dc->SQL = "AuthorById('267-41-2394')";

// Retrieve the record.

dc->Refresh();

// Use another Recordset as a convenience.

pRst = dc->GetRecordset();

printf("Author is %s %s", (LPSTR) (_bstr_t) pRst->Fields->
>GetItem("au_fname")->Value, \

(LPSTR) (_bstr_t) pRst->Fields->GetItem("au_lname")->Value);

pRst->Close();

} // End Try statement.

```

```

catch (_com_error &e)
{
PrintProviderError(pRst->GetActiveConnection());
PrintComError(e);
}
}

////////////////////////////////////

// //

// PrintProviderError Function //

// //

////////////////////////////////////

void PrintProviderError(_ConnectionPtr pConnection)
{
// Print Provider Errors from Connection object.
// pErr is a record object in the Connection's Error collection.
ErrorPtr pErr = NULL;
long nCount = 0;
long i = 0;
if( (pConnection->Errors->Count) > 0)
{
nCount = pConnection->Errors->Count;
// Collection ranges from 0 to nCount -1.
for(i = 0; i < nCount; i++)
{

```

```

pErr = pConnection->Errors->GetItem(i);
printf("\t Error number: %x\t%s", pErr->Number, pErr->Description);
}
}
}

////////////////////////////////////

//

// PrintComError Function //

// //

////////////////////////////////////

void PrintComError(_com_error &e)
{
_bstr_t bstrSource(e.Source());
_bstr_t bstrDescription(e.Description());

// Print Com errors.

printf("Error\n");
printf("\tCode = %08lx\n", e.Error());
printf("\tCode meaning = %s\n", e.ErrorMessage());
printf("\tSource = %s\n", (LPCSTR) bstrSource);
printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}

// EndHandlerCpp

```

See Also

[DataControl Object \(RDS\)](#) | [Handler Property \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

InternetTimeout Property Example (VC++)

This example demonstrates the [InternetTimeout](#) property, which exists on the [DataControl](#) and [DataSpace](#) objects. In this case, the **InternetTimeout** property is demonstrated on the **DataControl** object and the timeout is set to 20 seconds.

```
// BeginInternetTimeoutCpp

#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
no_namespace rename("EOF", "EndOfFile")

#import "C:\Program Files\Common Files\System\MSADC\msadco.dll"

#include <ole2.h>

#include <stdio.h>

#include <conio.h>

// Function declarations

inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};

void InternetTimeOutX(void);

void PrintProviderError(_ConnectionPtr pConnection);

void PrintComError(_com_error &e);

////////////////////////////////////

// //

// Main Function //

// //

////////////////////////////////////
```

```

void main()
{
if(FAILED(::CoInitialize(NULL)))
return;

InternetTimeOutX();

::CoUninitialize();
}

/////////////////////////////////////////////////////////////////

// //

// InternetTimeOutX Function //

// //

/////////////////////////////////////////////////////////////////

void InternetTimeOutX(void)
{
HRESULT hr = S_OK;

// Define ADO object pointers.
// Initialize pointers on define.
// These are in the ADODB:: namespace.
_RecordsetPtr pRst = NULL;

//Define RDS object pointers
RDS::IBindMgrPtr dc ;

try
{
TESTHR(dc.CreateInstance(__uuidof(RDS::DataControl)));
}
}

```

```
dc->Server = "http://MyServer";
dc->Connect = "Data Source='AuthorDatabase'";
dc->SQL = "SELECT * FROM Authors";
// Wait at least 20 seconds.
dc->InternetTimeout = 20000;
dc->Refresh();
// Use another Recordset as a convenience
pRst = dc->GetRecordset();
while(!(pRst->EndOfFile))
{
printf("%s %s", (LPSTR) (_bstr_t) pRst->Fields->
GetItem("au_fname")->Value,
(LPSTR) (_bstr_t) pRst->Fields->
GetItem("au_lname")->Value);
pRst->MoveNext();
}
pRst->Close();
}
catch (_com_error &e)
{
PrintProviderError(pRst->GetActiveConnection());
PrintComError(e);
}
}
```

```
////////////////////////////////////
// //
// PrintProviderError Function //
// //
////////////////////////////////////
void PrintProviderError(_ConnectionPtr pConnection)
{
// Print Provider Errors from Connection object.
// pErr is a record object in the Connection's Error collection.
ErrorPtr pErr = NULL;
if( (pConnection->Errors->Count) > 0)
{
long nCount = pConnection->Errors->Count;
// Collection ranges from 0 to nCount -1.
for(long i = 0; i < nCount; i++)
{
pErr = pConnection->Errors->GetItem(i);
printf("\t Error number: %x\t%s", pErr->Number,
pErr->Description);
}
}
}
////////////////////////////////////
// //
```

```

// PrintComError Function //
// //
////////////////////////////////////
void PrintComError(_com_error &e)
{
_bstr_t bstrSource(e.Source());
_bstr_t bstrDescription(e.Description());
// Print Com errors.
printf("Error\n");
printf("\tCode = %08lx\n", e.Error());
printf("\tCode meaning = %s\n", e.ErrorMessage());
printf("\tSource = %s\n", (LPCSTR) bstrSource);
printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}
// EndInternetTimeoutCpp

```

See Also

[InternetTimeoutProperty \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

RDS Code Examples in Microsoft Visual J++

Use the following code examples to learn how to use RDS properties when writing in Microsoft Visual J++.

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

- [Handler Property Example](#)
- [InternetTimeout Property Example](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

RDS 2.5 API Reference

Handler Property Example (VJ++)

This example demonstrates the [RDS DataControl](#) object [Handler](#) property. (See [DataFactory Customization](#) for more details.)

Assume the following sections in the parameter file, Msdfmap.ini, located on the server:

```
[connect AuthorDataBase]
Access=ReadWrite
Connect="DSN=Pubs"
[sql AuthorById]
SQL="SELECT * FROM Authors WHERE au_id = ?"
```

Your code looks like the following. The command assigned to the [SQL](#) property will match the *AuthorById* identifier and will retrieve a row for author Michael O'Leary. Although the [Connect](#) property in your code specifies the Northwind data source, that data source will be overwritten by the Msdfmap.ini *connect* section. The **DataControl** object's [Recordset](#) property is assigned to a disconnected [Recordset](#) object purely as a coding convenience.

```
// BeginHandlerJ

import com.ms.wfc.data.*;

import com.ms.wfc.data.rds.*;

import java.io.* ;

public class HandlerX

{

// The main entry point for the application.

public static void main (String[] args)

{

HandlerX();

System.exit(0);
```

```
}  
  
// HandlerX function  
static void HandlerX()  
{  
    // Define ADO Objects.  
    Recordset rstAuthors = null;  
  
    // Declarations.  
    BufferedReader in =  
        new BufferedReader (new InputStreamReader(System.in));  
    int intCount = 0;  
    int intDisplaysize = 15;  
    try  
    {  
        IBindMgr dc = (IBindMgr) new DataControl();  
        dc.setServer("MyServer");  
        dc.setConnect("Data Source=Northwind");  
        dc.setSQL("AuthorById(267-41-2394)");  
        dc.Refresh(); // Retrieve the record.  
        // Use another recordset as a convenience.  
        rstAuthors = (Recordset)dc.getRecordset();  
        System.out.println("Author is '" +  
            rstAuthors.getField("au_fname").getString() +  
            "' " +  
            rstAuthors.getField("au_lname").getString() +
```

```
""");
System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
// Notify user of any errors that result from ADO.
// As passing a Recordset, check for null pointer first.
if (rstAuthors != null)
{
PrintProviderError(rstAuthors.getActiveConnection());
}
else
{
System.out.println("Exception: " + ae.getMessage());
}
}
// System read requires this catch.
catch( java.io.IOException je)
{
PrintIOError(je);
}
catch(java.lang.UnsatisfiedLinkError e)
{
```

```
System.out.println("Exception: " + e.getMessage());
}
catch(java.lang.NullPointerException ne)
{
System.out.println(
"Exception: Attempt to use null where an object is required.");
}
finally
{
// Cleanup objects before exit.
if (rstAuthors != null)
if (rstAuthors.getState() == 1)
rstAuthors.close();
}
}
// PrintProviderError Function
static void PrintProviderError( Connection Cnn1 )
{
// Print Provider errors from Connection object.
// ErrItem is an item object in the Connection's Errors collection.
com.ms.wfc.data.Error ErrItem = null;
long nCount = 0;
int i = 0;
nCount = Cnn1.getErrors().getCount();
```

```

// If there are any errors in the collection, print them.
if( nCount > 0);
{
// Collection ranges from 0 to nCount - 1
for (i = 0; i< nCount; i++)
{
ErrItem = Cnn1.getErrors().getItem(i);
System.out.println("\t Error number: " + ErrItem.getNumber()
+ "\t" + ErrItem.getDescription() );
}
}
}

// PrintIOError Function
static void PrintIOError( java.io.IOException je)
{
System.out.println("Error \n");
System.out.println("\tSource = " + je.getClass() + "\n");
System.out.println("\tDescription = " + je.getMessage() + "\n");
}
}

// EndHandlerJ

```

See Also

[DataControl Object \(RDS\)](#) | [Handler Property \(RDS\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

RDS 2.5 API Reference

InternetTimeout Property Example (VJ++)

This example demonstrates the [InternetTimeout](#) property, which exists on the [DataControl](#) and [DataSpace](#) objects. In this case, the **InternetTimeout** property is demonstrated on the **DataControl** object and the timeout is set to 20 seconds.

```
// BeginInternetTimeoutJ

// The WFC class includes the ADO objects.

import com.ms.wfc.data.*;

import com.ms.wfc.data.rds.*;

import java.io.* ;

public class InternetTimeoutX

{

// The main entry point for the application.

public static void main (String[] args)

{

InternetTimeoutX();

System.exit(0);

}

// InternetTimeoutX function

static void InternetTimeoutX()

{

// Define ADO Objects.
```

```
Recordset rstAuthors = null;

// Declarations.

BufferedReader in =
new BufferedReader (new InputStreamReader(System.in));

int intCount = 0;

int intDisplaysize = 15;

try
{
IBindMgr dc = (IBindMgr) new DataControl();
dc.setServer("http://MyServer");
dc.setConnect("DSN=pubs");
dc.setSQL("SELECT * FROM Authors");
dc.setInternetTimeout(20000); // Wait at least 20 seconds.
dc.Refresh();

rstAuthors = (Recordset)dc.getRecordset();

while(!rstAuthors.getEOF())
{
System.out.println(rstAuthors.getField
("au_fname").getString() + " " +
rstAuthors.getField("au_lname").getString());

intCount++;

if(intCount % intDisplaysize == 0)
{
System.out.println("\nPress <Enter> to continue..");
```

```
in.readLine();
intCount = 0;
}
rstAuthors.moveNext();
}
System.out.println("\nPress <Enter> to continue..");
in.readLine();
}
catch( AdoException ae )
{
// Notify user of any errors that result from ADO.
// As passing a Recordset, check for null pointer first.
if (rstAuthors != null)
{
PrintProviderError(rstAuthors.getActiveConnection());
}
else
{
System.out.println("Exception: " + ae.getMessage());
}
}
// System read requires this catch.
catch( java.io.IOException je)
{
```

```
PrintIOError(je);
}
catch(java.lang.UnsatisfiedLinkError e)
{
System.out.println("Exception: " + e.getMessage());
}
catch(java.lang.NullPointerException ne)
{
System.out.println(
"Exception: Attempt to use null where an object is required.");
}
finally
{
// Cleanup objects before exit.
if (rstAuthors != null)
if (rstAuthors.getState() == 1)
rstAuthors.close();
}
}
// PrintProviderError Function
static void PrintProviderError( Connection Cnn1 )
{
// Print Provider errors from Connection object.
// ErrItem is an item object in the Connection's Errors collection.
```

```
com.ms.wfc.data.Error ErrItem = null;

long nCount = 0;

int i = 0;

nCount = Cnn1.getErrors().getCount();

// If there are any errors in the collection, print them.

if( nCount > 0);

{

// Collection ranges from 0 to nCount - 1

for (i = 0; i< nCount; i++)

{

ErrItem = Cnn1.getErrors().getItem(i);

System.out.println("\t Error number: " + ErrItem.getNumber()

+ "\t" + ErrItem.getDescription() );

}

}

}

// PrintIOError Function

static void PrintIOError( java.io.IOException je)

{

System.out.println("Error \n");

System.out.println("\tSource = " + je.getClass() + "\n");

System.out.println("\tDescription = " + je.getMessage() + "\n");

}

}
```

```
// EndInternetTimeoutJ
```

See Also

[InternetTimeout Property \(RDS\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

ADO MD API Reference

This section of the ADO MD documentation contains topics for each ADO MD object, collection, method, and property, as well as example code when appropriate. For more information, search for a specific topic in the index or refer to the following topics:

- [ADO MD Objects](#)
- [ADO MD Collections](#)
- [ADO MD Properties](#)
- [ADO MD Methods](#)
- [ADO MD Enumerated Constants](#)
- [ADO MD Examples](#)

See Also

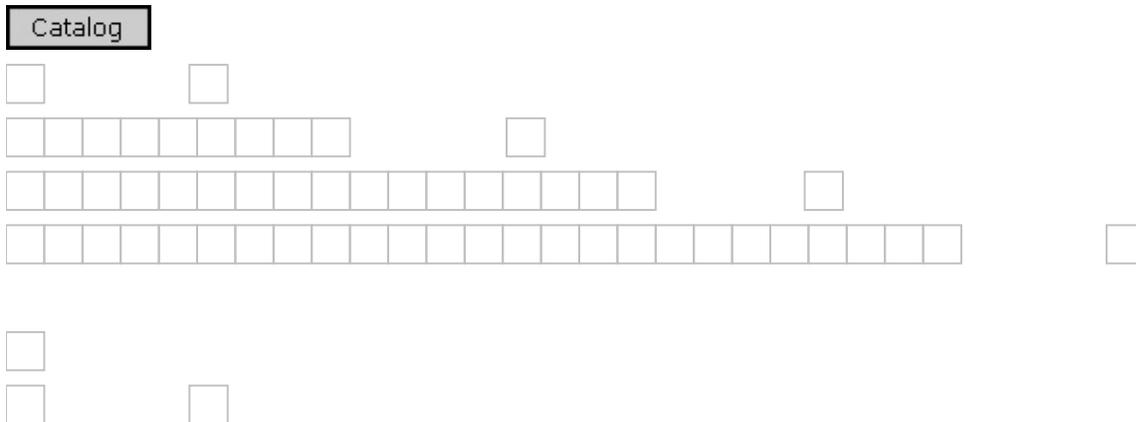
[ADO MD Code Examples](#) | [ADO MD Collections](#) | [ADO MD Enumerated Constants](#) | [ADO MD Methods](#) | [ADO MD Object Model](#) | [ADO MD Objects](#) | [ADO MD Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

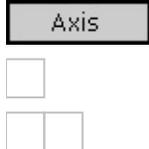
ADO 2.5 MD API Reference

ADO MD Object Model

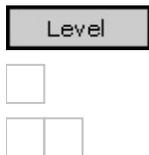
The following diagram shows how these objects are represented and related in ADO MD.



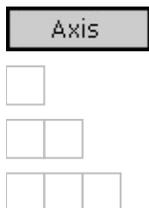
The [Axis](#) and [Cell](#) objects each have a [Positions](#) collection.

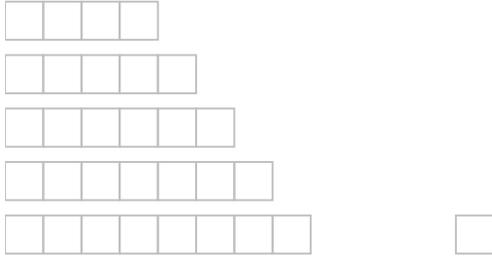


The [Level](#) and [Position](#) objects each have a [Members](#) collection.



The [Axis](#), [Cell](#), [Cellset](#), [CubeDef](#), [Dimension](#), [Hierarchy](#), [Level](#), and [Member](#) objects each have a standard ADO [Properties](#) collection.





See Also

[ADO MD API Reference](#) | [ADO MD Code Examples](#) | [ADO MD Collections](#) | [ADO MD Enumerated Constants](#) | [ADO MD Methods](#) | [ADO MD Objects](#) | [ADO MD Properties](#) | [Microsoft ADO MD Programmer's Reference](#) | [Overview of Multidimensional Schemas and Data](#) | [Working with Multidimensional Data](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 MD API Reference

ADO MD Objects

Axis	Represents a positional or filter axis of a cellset, containing selected members of one or more dimensions.
Catalog	Contains multidimensional schema information (that is, cubes and underlying dimensions, hierarchies , levels, and members) specific to a multidimensional data provider (MDP).
Cell	Represents the data at the intersection of axis coordinates, contained in a cellset.
Cellset	Represents the results of a multidimensional query. It is a collection of cells selected from cubes or other cellsets.
CubeDef	Represents a cube from a multidimensional schema, containing a set of related dimensions.
Dimension	Represents one of the dimensions of a multidimensional cube, containing one or more hierarchies of members.
Hierarchy	Represents one way in which the members of a dimension can be aggregated or "rolled up." A dimension can be aggregated along one or more hierarchies.
Level	Contains a set of members, each of which has the same rank within a hierarchy.
Member	Represents a member of a level in a cube, the children of a member of a level, or a member of a position along an axis of a cellset.
Position	Represents a set of one or more members of different dimensions that defines a point along an axis.

Also, the **Catalog** object is connected to an ADO **Connection** object, which is included with the standard ADO library:

Object	Description
Connection	Represents an open connection to a data source.

The relationships between these objects are illustrated in the [ADO MD Object Model](#).

Many ADO MD objects can be contained in a corresponding collection. For example, a [CubeDef](#) object can be contained in a [CubeDefs](#) collection of a **Catalog**. For more information, see [ADO MD Collections](#).

See Also

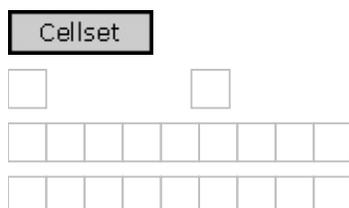
[ADO MD API Reference](#) | [ADO MD Code Examples](#) | [ADO MD Collections](#) | [ADO MD Enumerated Constants](#) | [ADO MD Methods](#) | [ADO MD Object Model](#) | [ADO MD Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Axis Object

Represents a positional or filter axis of a cellset, containing selected members of one or more dimensions.



Remarks

An **Axis** object can be contained by an [Axes](#) collection, or returned by the [FilterAxis](#) property of a [Cellset](#).

With the collections and properties of an **Axis** object, you can do the following:

- Identify the **Axis** with the [Name](#) property.
- Iterate through each position along an **Axis** using the [Positions](#) collection.
- Obtain the number of dimensions on the **Axis** with the [DimensionCount](#) property.
- Obtain [provider](#)-specific attributes of the **Axis** with the standard ADO [Properties](#) collection.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#) | [Axes Collection](#) | [Positions Collection](#) | [Properties Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 MD API Reference

Axis Object Properties, Methods, and Events

Properties/Collections

[DimensionCount Property](#)

[Name Property](#)

[Positions Collection](#)

[Properties Collection](#)

Methods

None.

Events

None.

See Also

Applies To: [Axis Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Catalog Object

Contains multidimensional schema information (that is, cubes and underlying dimensions, [hierarchies](#), levels, and members) specific to a multidimensional [data provider](#) (MDP).

Catalog



Remarks

With the collections and properties of a **Catalog** object, you can do the following:

- Open the catalog by setting the [ActiveConnection](#) property to a standard ADO [Connection](#) object or to a valid connection string.
- Identify the **Catalog** with the [Name](#) property.
- Iterate through the cubes in a catalog using the [CubeDefs](#) collection.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [Connection Object](#) | [CubeDefs Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Catalog Object Properties, Methods, and Events

Properties/Collections

[ActiveConnection Property](#)

[CubeDefs Collection](#)

[Name Property](#)

Methods

None.

Events

None.

See Also

Applies To: [Catalog Object](#)

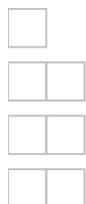
[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Cell Object

Represents the data at the intersection of axis coordinates contained in a cellset.

Cellset



Remarks

A **Cell** object is returned by the [Item](#) property of a [Cellset](#) object.

With the collections and properties of a **Cell** object, you can do the following:

- Return the data in the **Cell** with the [Value](#) property.
- Return the string representing the formatted display of the **Value** property with the [FormattedValue](#) property.
- Return the ordinal value of the **Cell** within the **Cellset** with the [Ordinal](#) property.
- Determine the position of the **Cell** within the [CubeDef](#) with the [Positions](#) collection.
- Retrieve other information about the **Cell** with the standard ADO [Properties](#) collection.

The **Properties** collection contains [provider](#)-supplied properties. The following table lists properties that might be available. The actual property list may differ depending upon the implementation of the provider. See the documentation for your provider for a more complete list of available properties.

Name	Description
BackColor	Background color used when displaying the cell.
FontFlags	Bitmask detailing effects on the font.
FontName	Font used to display the cell value.

FontSize Font size used to display the cell value.
ForeColor Foreground color used when displaying the cell.
FormatString Value in a formatted string.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#) | [Cellset Object](#) | [Positions Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Cell Object Properties, Methods, and Events

Properties/Collections

[FormattedValue Property](#)

[Ordinal Property \(Cell\)](#)

[Positions Collection](#)

[Properties Collection](#)

[Value Property](#)

Methods

None.

Events

None.

See Also

Applies To: [Cell Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Cellset Object

Represents the results of a multidimensional query. It is a collection of cells selected from cubes or other cellsets.

Cellset



Remarks

Data within a **Cellset** is retrieved using direct, array-like access. You can "drill down" to a specific member to obtain data about that member. For example, the following code returns the caption of the first member in the first position on the first axis of a cellset named cst:

```
cst.Axes(0).Positions(0).Members(0).Caption
```

There is no notion of a current cell within a cellset. Instead, the [Item](#) property retrieves a specific [Cell](#) object from the cellset. The arguments of the **Item** property determine which cell is retrieved. You can specify the unique ordinal value of a cell. You can also retrieve cells by using their position numbers along each axis of the cellset. For more information about retrieving cells, see the [Item](#) property.

With the collections, methods, and properties of a **Cellset** object, you can do the following:

- Associate an open connection with a **Cellset** object by setting its [ActiveConnection](#) property.
- Execute and retrieve the results of a multidimensional query with the [Open](#) method.
- Retrieve a **Cell** from the **Cellset** with the [Item](#) property.
- Return the [Axis](#) objects that define the **Cellset** with the [Axes](#) collection.
- Retrieve information about the dimensions used to filter the data in the

- Cellset** with the [FilterAxis](#) property.
- Return or specify the query used to define the **Cellset** with the [Source](#) property.
 - Return the current state of the **Cellset** (open, closed, executing, or connecting) with the [State](#) property.
 - Close an open **Cellset** with the [Close](#) method.
 - Retrieve [provider](#)-specific information about the **Cellset** with the standard ADO [Properties](#) collection.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [Axes Collection](#) | [Cell Object](#) | [Connection Object](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Cellset Object Properties, Methods, and Events

Properties/Collections

[ActiveConnection Property](#)

[Axes Collection](#)

[FilterAxis Property](#)

[Item Property \(Cellset\)](#)

[Properties Collection](#)

[Source Property](#)

[State Property](#)

Methods

[Close Method](#)

[Open Method](#)

Events

None.

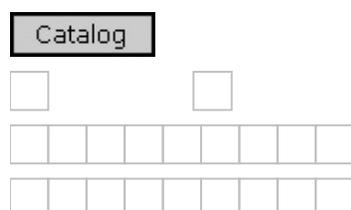
See Also

Applies To: [Cellset Object](#)

ADO 2.5 MD API Reference

CubeDef Object

Represents a cube from a multidimensional schema, containing a set of related dimensions.



Remarks

With the collections and properties of a **CubeDef** object, you can do the following:

- Identify a **CubeDef** with the [Name](#) property.
- Return a string that describes the cube with the [Description](#) property.
- Return the dimensions that make up the cube with the [Dimensions](#) collection.
- Obtain additional information about the **CubeDef** with the standard ADO [Properties](#) collection.

The **Properties** collection contains provider-supplied properties. The following table lists properties that might be available. The actual property list may differ depending upon the implementation of the [provider](#). See the documentation for your provider for a more complete list of available properties.

Name	Description
CatalogName	The name of the catalog to which this cube belongs.
CreatedOn	Date and time of cube creation.
CubeGUID	Cube GUID.
CubeName	The name of the cube.
CubeType	The type of the cube.
DataUpdatedBy	User ID of the person doing the last data update.

Description	A meaningful description of the cube.
LastSchemaUpdate	Date and time of last schema update.
SchemaName	The name of the schema to which this cube belongs.
SchemaUpdatedBy	User ID of the person doing the last schema update.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#) | [Catalog Object](#) | [CubeDefs Collection](#) | [Dimensions Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

CubeDef Object Properties, Methods, and Events

Properties/Collections

[Description Property](#)

[Dimensions Collection](#)

[Name Property](#)

[Properties Collection](#)

Events

None.

See Also

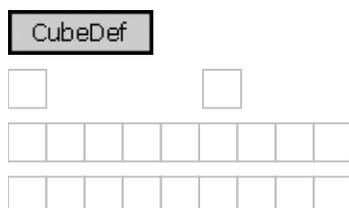
Applies To: [CubeDef Object](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Dimension Object

Represents one of the dimensions of a multidimensional cube, containing one or more [hierarchies](#) of members.



Remarks

With the collections and properties of a **Dimension** object, you can do the following:

- Identify the **Dimension** with the [Name](#) and [UniqueName](#) properties.
- Return a meaningful string that describes the **Dimension** with the [Description](#) property.
- Return the [Hierarchy](#) objects that make up the **Dimension** with the [Hierarchies](#) collection.
- Use the standard ADO [Properties](#) collection to obtain additional information about the **Dimension** object.

The **Properties** collection contains provider-supplied properties. The following table lists properties that might be available. The actual property list may differ depending upon the implementation of the [provider](#). See the documentation for your provider for a more complete list of available properties.

Name	Description
CatalogName	The name of the catalog to which this cube belongs.
CubeName	The name of the cube.
DefaultHierarchy	The unique name of the default hierarchy .
Description	A meaningful description of the cube.
DimensionCaption	A label or caption associated with the dimension .

DimensionCardinality	The number of members in the dimension.
DimensionGUID	The GUID of the dimension.
DimensionName	The name of the dimension.
DimensionOrdinal	The ordinal number of the dimension among the group of dimensions that form the cube.
DimensionType	The dimension type.
DimensionUniqueName	The unambiguous name of the dimension.
SchemaName	The name of the schema to which this cube belongs.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#) | [CubeDef Object](#) | [Dimensions Collection](#) | [Hierarchies Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Dimension Object Properties, Methods, and Events

Properties/Collections

[Description Property](#)

[Hierarchies Collection](#)

[Name Property](#)

[Properties Collection](#)

[UniqueName Property](#)

Methods

None.

Events

None.

See Also

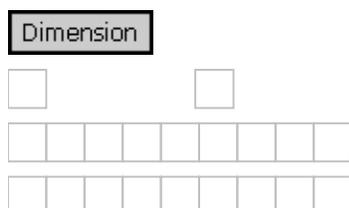
Applies To: [Dimension Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Hierarchy Object

Represents one way in which the members of a [dimension](#) can be aggregated or "rolled up." A dimension can be aggregated along one or more [hierarchies](#).



Remarks

With the collections and properties of a **Hierarchy** object, you can do the following:

- Identify the **Hierarchy** with the [Name](#) and [UniqueName](#) properties.
- Return a meaningful string that describes the **Hierarchy** with the [Description](#) property.
- Return the [Level](#) objects that make up the **Hierarchy** with the [Levels](#) collection.
- Use the standard ADO [Properties](#) collection to obtain additional information about the **Hierarchy** object.

The **Properties** collection contains provider-supplied properties. The following table lists properties that might be available. The actual property list may differ depending upon the implementation of the [provider](#). See the documentation for your provider for a more complete list of available properties.

Name	Description
AllMember	The member at the highest level of rollup in the hierarchy.
CatalogName	The name of the catalog to which this cube belongs.
CubeName	The name of the cube.
DefaultMember	The unique name of the default member for this hierarchy.

Description	A meaningful description of the hierarchy.
DimensionType	The type of dimension to which this hierarchy belongs.
DimensionUniqueName	The unambiguous name of the dimension.
HierarchyCaption	A label or caption associated with the hierarchy.
HierarchyCardinality	The number of members in the hierarchy.
HierarchyGUID	The GUID of the hierarchy.
HierarchyName	The name of the hierarchy.
HierarchyUniqueName	The unambiguous name of the hierarchy.
SchemaName	The name of the schema to which this cube belongs.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#) | [Dimension Object](#) | [Hierarchies Collection](#) | [Levels Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Hierarchy Object Properties, Methods, and Events

Properties/Collections

[Description Property](#)

[Levels Collection](#)

[Name Property](#)

[Properties Collection](#)

[UniqueName Property](#)

Methods

None.

Events

None.

See Also

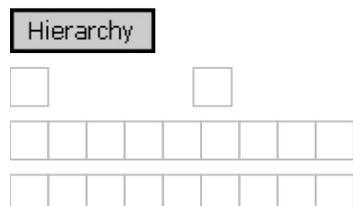
Applies To: [Hierarchy Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Level Object

Contains a set of members, each of which has the same rank within a [hierarchy](#).



Remarks

With the collections and properties of a **Level** object, you can do the following:

- Identify the **Level** with the [Name](#) and [UniqueName](#) properties.
- Return a string to use when displaying the **Level** with the [Caption](#) property.
- Return a meaningful string that describes the **Level** with the [Description](#) property.
- Return the [Member](#) objects that make up the **Level** with the [Members](#) collection.
- Return the number of levels from the root of the **Level** with the [Depth](#) property.
- Use the standard ADO [Properties](#) collection to obtain additional information about the **Level** object.

The **Properties** collection contains provider-supplied properties. The following table lists properties that might be available. The actual property list may differ depending upon the implementation of the [provider](#). See the documentation for your provider for a more complete list of available properties.

Name	Description
CatalogName	The name of the catalog to which this cube belongs.
CubeName	The name of the cube.
Description	A meaningful description of the level.
DimensionUniqueName	The unambiguous name of the dimension .
HierarchyUniqueName	The unambiguous name of the hierarchy .

LevelCaption	A label or caption associated with the level.
LevelCardinality	The number of members in the level.
LevelGUID	The GUID of the level.
LevelName	Name of the level.
LevelNumber	The distance between the level and the root of the hierarchy.
LevelType	The type of level.
LevelUniqueName	The unambiguous name of the level.
SchemaName	The name of the schema to which this cube belongs.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#) | [Hierarchy Object](#) | [Levels Collection](#) | [Members Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Level Object Properties, Methods, and Events

Properties/Collections

[Caption Property](#)

[Depth Property](#)

[Description Property](#)

[Members Collection](#)

[Name Property](#)

[Properties Collection](#)

[UniqueName Property](#)

Methods

None.

Events

None.

See Also

Applies To: [Level Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Member Object

Represents a member of a level in a cube, the children of a [member](#) of a level, or a member of a position along an axis of a cellset.



Remarks

The properties of a **Member** differ depending on the context in which it is used. A **Member** of a [Level](#) in a [CubeDef](#) has a [Children](#) property that returns the **Members** on the next lower level in the hierarchy from the current **Member**. For a **Member** of a [Position](#), the **Children** collection is always empty. Also, the [Type](#) property applies only to **Members** of a **Level**.

A **Member** of **Position** has two properties—[DrilledDown](#) and [ParentSameAsPrev](#)—that are useful when displaying the [Cellset](#). An error will occur if these properties are accessed on a **Member** of a **Level**.

With the collections and properties of a **Member** object of a **Level**, you can do the following:

- Identify the **Member** with the [Name](#) and [UniqueName](#) properties.
- Return a string to use when displaying the **Member** with the [Caption](#) property.
- Return a meaningful string that describes a measure or formula **Member** with the [Description](#) property.
- Determine the nature of the **Member** with the [Type](#) property.
- Obtain information about the **Level** of the **Member** with the [LevelDepth](#) and [LevelName](#) properties.
- Obtain related **Members** in a [Hierarchy](#) with the [Parent](#) and [Children](#) properties.
- Count the children of a **Member** with the [ChildCount](#) property.

- Use the standard ADO [Properties](#) collection to obtain additional information about the **Level** object.

With the collections and properties of a **Member** of a **Position** along an [Axis](#), you can do the following:

- Identify the **Member** with the [Name](#) and [UniqueName](#) properties.
- Return a string to use when displaying the **Member** with the [Caption](#) property.
- Return a meaningful string that describes a measure or formula **Member** with the [Description](#) property.
- Obtain information about the **Level** of the **Member** with the [LevelDepth](#) and [LevelName](#) properties.
- Count the [children](#) of a **Member** with the [ChildCount](#) property.
- Use the [DrilledDown](#) property to determine whether there is at least one child on the **Axis** immediately following this **Member**.
- Use the [ParentSameAsPrev](#) property to determine whether the parent of this **Member** is the same as the [parent](#) of the immediately preceding **Member**.
- Use the standard ADO [Properties](#) collection to obtain additional information about the **Level** object.

The **Properties** collection contains provider-supplied properties. The following table lists properties that might be available. The actual property list may differ depending upon the implementation of the [provider](#). See the documentation for your provider for a more complete list of available properties.

Name	Description
CatalogName	The name of the catalog to which this cube belongs.
ChildrenCardinality	The number of children that the member has.
CubeName	The name of the cube.
Description	A meaningful description of the member .
DimensionUniqueName	The unambiguous name of the dimension .
HierarchyUniqueName	The unambiguous name of the hierarchy .
LevelNumber	The distance between the level and the root of the hierarchy.
LevelUniqueName	The unambiguous name of the level.
MemberCaption	A label or caption associated with the member.

MemberGUID	The GUID of the member.
MemberName	The name of the member.
MemberOrdinal	The ordinal number of the member.
MemberType	The type of the member.
MemberUniqueName	The unambiguous name of the member.
ParentCount	The count of the number of parents that this member has.
ParentLevel	The level number of the member's parent.
ParentUniqueName	The unambiguous name of the member's parent.
SchemaName	The name of the schema to which this cube belongs.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [Members Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Member Object Properties, Methods, and Events

Properties/Collections

[Caption Property](#)

[ChildCount Property](#)

[Children Property](#)

[Description Property](#)

[DrilledDown Property](#)

[LevelDepth Property](#)

[LevelName Property](#)

[Name Property](#)

[Parent Property](#)

[ParentSameAsPrev Property](#)

[Properties Collection](#)

[Type Property](#)

[UniqueName Property](#)

Methods

None.

Events

None.

See Also

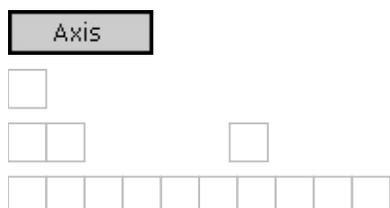
Applies To: [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Position Object

Represents a set of one or more members of different dimensions that defines a point along an axis.



Remarks

With the properties and collections of a **Position** object you can do the following:

- Use the **Ordinal** property to return the ordinal position of the **Position** along the [Axis](#).
- Use the [Members](#) collection to return the members that make up the position along the **Axis**.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#) | [Axis Object](#) | [Cell Object](#) | [Members Collection](#) | [Positions Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 MD API Reference

Position Object Properties, Methods, and Events

Properties/Collections

[Members Collection](#)

[Ordinal Property \(Position\)](#)

Methods

None.

Events

None.

See Also

Applies To: [Position Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

ADO MD Collections

Axes	Contains the Axis objects that define a cellset.
CubeDefs	Contains the CubeDef objects that represent a cube from a multidimensional catalog.
Dimensions	Contains the Dimension objects that make up a cube.
Hierarchies	Contains the set Hierarchy objects from a dimension.
Levels	Contains the Level objects that make up a hierarchy .
Members	Contains the Member objects from a level or a position along an axis.
Positions	Contains the Position objects that define a point on an axis.

See Also

[ADO MD API Reference](#) | [ADO MD Code Examples](#) | [ADO MD Enumerated Constants](#) | [ADO MD Methods](#) | [ADO MD Object Model](#) | [ADO MD Objects](#) | [ADO MD Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Axes Collection

Contains the [Axis](#) objects that define a cellset.

Cellset



Remarks

A [Cellset](#) object contains an **Axes** collection. Once the **Cellset** is opened, this collection will contain at least one **Axis**. See the [Axis](#) object for a more detailed explanation of how to use **Axis** objects.

Note The filter axis of a **Cellset** is not contained in the **Axes** collection. See the [FilterAxis](#) property for more information.

Axes is a standard ADO collection. With the properties and methods of a collection, you can do the following:

- Obtain the number of objects in the collection with the [Count](#) property.
- Return an object from the collection with the default [Item](#) property.
- Update the objects in the collection from the [provider](#) with the [Refresh](#) method.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [Axis Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Axes Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Refresh Method](#)

Events

None.

See Also

Applies To: [Axes Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

CubeDefs Collection

Contains the [CubeDef](#) objects that represent a cube from a multidimensional catalog.



Remarks

CubeDefs is a standard ADO collection. With the properties and methods of a collection, you can do the following:

- Obtain the number of objects in the collection with the [Count](#) property.
- Return an object from the collection with the default [Item](#) property.
- Update the objects in the collection from the [provider](#) with the [Refresh](#) method.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [Catalog Object](#) | [CubeDef Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

CubeDefs Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Refresh Method](#)

Events

None.

See Also

Applies To: [CubeDefs Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Dimensions Collection

Contains the [Dimension](#) objects that make up a cube.

CubeDef



Remarks

Dimensions is a standard ADO collection. With the properties and methods of a collection, you can do the following:

- Obtain the number of objects in the collection with the [Count](#) property.
- Return an object from the collection with the default [Item](#) property.
- Update the objects in the collection from the provider with the [Refresh](#) method.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [CubeDef Object](#) | [Dimension Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Dimensions Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Refresh Method](#)

Events

None.

See Also

Applies To: [Dimensions Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Hierarchies Collection

Contains the set [Hierarchy](#) objects from a dimension.

Dimension



Remarks

Hierarchies is a standard ADO collection. With the properties and methods of a collection, you can do the following:

- Obtain the number of objects in the collection with the [Count](#) property.
- Return an object from the collection with the default [Item](#) property.
- Update the objects in the collection from the [provider](#) with the [Refresh](#) method.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [Dimension Object](#) | [Hierarchy Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Hierarchies Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Refresh Method](#)

Events

None.

See Also

Applies To: [Hierarchies Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Levels Collection

Contains the [Level](#) objects that make up a [hierarchy](#).

Hierarchy



Remarks

Levels is a standard ADO collection. With the properties and methods of a collection, you can do the following:

- Obtain the number of objects in the collection with the [Count](#) property.
- Return an object from the collection with the default [Item](#) property.
- Update the objects in the collection from the [provider](#) with the [Refresh](#) method.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [Hierarchy Object](#) | [Level Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Levels Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Refresh Method](#)

Events

None.

See Also

Applies To: [Levels Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Members Collection

Contains the [Member](#) objects from a level or a position along an axis.

Level



Remarks

A **Members** collection is used to contain the following types of members:

- The members that make up a level in a cube. These are contained in the **Members** collection of a [Level](#) object. For example, using the sample from [Overview of Multidimensional Schemas and Data](#), the four members of the Countries level are Canada, USA, UK, and Germany.
- The members that are the [children](#) of a specific member within a hierarchy. These members are returned by the [Children](#) property of the [parent Member](#) object. For example, again using the same sample, the two children of the Canada member are Canada-East and Canada-West.
- The members that define a specific position along an axis of a [cellset](#). Using the cellset from [Working with Multidimensional Data](#) as an example, the two members of the first position on the x-axis are Valentine and Seattle. These members are contained by the **Members** collection of a [Position](#) object.

Members is a standard ADO collection. With the properties and methods of a collection, you can do the following:

- Obtain the number of objects in the collection with the [Count](#) property.
- Return an object from the collection with the default [Item](#) property.
- Update the objects in the collection from the [provider](#) with the [Refresh](#) method.

See Also

[VBScript Example](#)

[Properties, Methods, and Events](#) | [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Members Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Refresh Method](#)

Events

None.

See Also

Applies To: [Members Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Positions Collection

Contains the [Position](#) objects that define a point on an axis.



Remarks

Positions is a standard ADO collection. With the properties and methods of a collection, you can do the following:

- Obtain the number of objects in the collection with the [Count](#) property.
- Return an object from the collection with the default [Item](#) property.
- Update the objects in the collection from the [provider](#) with the [Refresh](#) method.

See Also

[Visual Basic Example](#)

[Properties, Methods, and Events](#) | [Axis Object](#) | [Cell Object](#) | [Position Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Positions Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Refresh Method](#)

Events

None.

See Also

Applies To: [Positions Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

ADO MD Properties

ActiveConnection	Indicates to which ADO Connection object the current cellset or catalog currently belongs.
Caption	Indicates the text caption to use when displaying a Level or Member object.
ChildCount	Indicates the number of members for which the current Member object is the parent in a hierarchy .
Children	Returns a collection of Members for which the current Member is the parent in the hierarchy.
Count	Indicates the number of objects in a collection.
Depth	Indicates the number of levels between the Level and the root of the hierarchy level.
Description	Returns a text explanation of the current object.
DimensionCount	Indicates the number of dimensions on an axis.
DrilledDown	Indicates whether no children immediately follow the member on the axis.
FilterAxis	Indicates filter information for the current cellset.
Item	Retrieves a cell from a cellset using its coordinates.
Item	Returns a specific member of a collection by name or ordinal number.
FormattedValue	Indicates the formatted display of a cell value.
LevelDepth	Indicates the number of levels between the root of the hierarchy and a member.
LevelName	Indicates the name of the level of a member.
Name	Indicates the name of an object.
Ordinal (Cell)	Uniquely identifies a cell by its position within a cellset.
Ordinal (Position)	Uniquely identifies a position along an axis.
Parent	Indicates the member that is the parent of the current member in a hierarchy.
ParentSameAsPrev	Indicates whether the parent of this position member is the same as the parent of the immediately preceding member.
Source	Indicates the source for the data in the cellset.

State	Indicates the current state of the cellset.
Type	Indicates the type of the current member.
UniqueName	Indicates an unambiguous name for the current object.
Value	Indicates the value of the current cell.

See Also

[ADO MD API Reference](#) | [ADO MD Code Examples](#) | [ADO MD Collections](#) | [ADO MD Enumerated Constants](#) | [ADO MD Methods](#) | [ADO MD Object Model](#) | [ADO MD Objects](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

ActiveConnection Property

Indicates to which ADO [Connection](#) object the current cellset or catalog currently belongs.

Settings and Return Values

Sets or returns a **Variant** that contains a string defining a connection or **Connection** object. The default is empty.

Remarks

You can set this property to a valid ADO **Connection** object or to a valid connection string. When this property is set to a connection string, the provider creates a new **Connection** object using this definition and opens the connection.

If you use the **ActiveConnection** argument of the [Open](#) method to open a [Cellset](#) object, the **ActiveConnection** property will inherit the value of the argument.

Setting the **ActiveConnection** property of a [Catalog](#) object to **Nothing** releases the associated data, including data in the [CubeDefs](#) collection and any related [Dimension](#), [Hierarchy](#), [Level](#), and [Member](#) objects. Closing a **Connection** object that was used to open a **Catalog** has the same effect as setting the **ActiveConnection** property to **Nothing**.

Changing the default database of the connection referenced by the **ActiveConnection** property of a **Catalog** object invalidates the contents of the **Catalog**.

An error will occur if you attempt to change the **ActiveConnection** property for an open **Cellset** object.

Note In Visual Basic, remember to use the **Set** keyword when setting the **ActiveConnection** property to a **Connection** object. If you omit the **Set** keyword, you will actually be setting the **ActiveConnection** property equal to the **Connection** object's default property, **ConnectionString**. The code

will work; however, you will create an additional connection to the data source, which may have negative performance implications.

When using the MSOLAP data provider, set the data source in a connection string to a server name and set the initial catalog to the name of a catalog from the data source. To connect to a cube file that is disconnected from a server, set the location to the full path to the .CUB file. In either case, set the [provider](#) to the provider name. For example, the following string connects to a catalog named Bobs Video Store on a server named Servername with the MSOLAP Provider:

```
"Data Source=Servername;Initial Catalog=Bobs Video Store;Provider=ms
```

The following string connects to a local cube file at the location C:\MSDASDK\samples\oledb\olap\data\bobsvid.cub:

```
"Location=C:\MSDASDK\samples\oledb\olap\data\bobsvid.cub;Provider=ms
```

See Also

[Visual Basic Example](#)

[Connection Object](#) | [Open Method](#)

Applies To: [Catalog Object](#) | [Cellset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Caption Property

Indicates the text caption to use when displaying a [Level](#) or [Member](#) object.

Return Values

Returns a **String** and is read-only.

See Also

[Visual Basic Example](#)

[Description Property](#)

Applies To: [Level Object](#) | [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

ChildCount Property

Indicates the number of members for which the current [Member](#) object is the [parent](#) in a [hierarchy](#).

Return Values

Returns a **Long** integer and is read-only.

Remarks

Use the **ChildCount** property to return an estimate of how many [children](#) a **Member** has. The actual children of a **Member** can be returned by the [Children](#) property.

For **Member** objects from a [Position](#) object, the maximum number returned is 65536. If the actual number of children exceeds 65536, the value returned will still be 65536. Therefore, the application should interpret a **ChildCount** of 65536 as equal to or greater than 65536 children.

For **Member** objects from a [Level](#) object, use the ADO collection [Count](#) property on the **Children** collection to determine the exact number of children. Determining the exact number of children may be slow if the number of children in the collection is large.

See Also

[Children Property](#) | [Count Property](#) | [Members Collection](#)

Applies To: [Member Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADO 2.5 MD API Reference

Children Property

Returns a [Members](#) collection for which the current [Member](#) is the [parent](#) in the [hierarchy](#).

Return Values

Returns a **Members** collection and is read-only.

Remarks

The **Children** property contains a **Members** collection for which the current **Member** is the hierarchical parent. Leaf level **Member** objects have no [child](#) members in the **Members** collection. This property is only supported on **Member** objects belonging to a [Level](#) object. An error occurs when this property is referenced from **Member** objects belonging to a [Position](#) object.

See Also

[ChildCount Property](#)

Applies To: [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Depth Property

Indicates the number of levels between the [Level](#) and the root of the [hierarchy](#).

Return Values

Returns an integer, and is read-only.

Remarks

A **Level** at the root of a hierarchy has a **Depth** value of zero (0).

See Also

[LevelDepth Property](#)

Applies To: [Level Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Description Property

Returns a text explanation of the current object.

Return Values

Returns a **String** and is read-only.

Remarks

For [Member](#) objects, **Description** applies only to measure and formula members. **Description** returns an empty string ("") for all other types of members. For more information about the various types of members, see the [Type](#) property.

This property is only supported on **Member** objects belonging to a [Level](#) object. An error occurs when this property is referenced from **Member** objects belonging to a [Position](#) object.

See Also

Applies To: [CubeDef Object](#) | [Dimension Object](#) | [Hierarchy Object](#) | [Level Object](#) | [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

DimensionCount Property

Indicates the number of dimensions on an axis.

Return Values

Returns a **Long** integer, and is read-only.

See Also

[VBScript Example](#)

[Dimension Object](#)

Applies To: [Axis Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

DrilledDown Property

Indicates whether no children immediately follow the member on the axis.

Return Values

Returns a **Boolean** value and is read-only. **DrilledDown** returns **True** if there are no [child](#) members of the current member on the axis. **DrilledDown** returns **False** if there is one or more child members of the current member on the axis.

Remarks

Use the **DrilledDown** property to determine whether there is at least one child of this member on the axis immediately following this member. This information is useful when displaying the member.

This property is only supported on [Member](#) objects belonging to a [Position](#) object. An error occurs when this property is referenced from **Member** objects belonging to a [Level](#) object.

See Also

[ParentSameAsPrev Property](#)

Applies To: [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

FilterAxis Property

Indicates filter information about the current cellset.

Return Values

Returns an [Axis](#) object, and is read-only.

Remarks

Use the **FilterAxis** property to return information about the dimensions that were used to slice the data. The [DimensionCount](#) property of the **Axis** returns the number of slicer dimensions. This axis usually has just one row.

The **Axis** returned by [FilterAxis](#) is not contained in the [Axes](#) collection for a [Cellset](#) object.

See Also

[Axis Object](#) | [Dimension Object](#) | [DimensionCount Property](#)

Applies To: [Cellset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

FormattedValue Property

Indicates the formatted display of a cell value.

Return Values

Returns a **String** and is read-only.

Remarks

Use the **FormattedValue** property to obtain the formatted display value of the [Value](#) property of a [Cell](#) object. For example, if the value of a cell was 1056.87, and this value represented a dollar amount, **FormattedValue** would be \$1,056.87.

See Also

[Visual Basic Example](#)

[Value Property](#)

Applies To: [Cell Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Item Property (Cellset)

Retrieves a cell from a cellset using its coordinates.

Syntax

```
Set Cell = Cellset.Item ( Positions )
```

Parameters

Positions

A **Variant Array** of values that uniquely specify a cell. *Positions* can be one of the following:

- An array of position numbers
- An array of member names
- The ordinal position

Remarks

Use the **Item** property to return a [Cell](#) object within a [Cellset](#) object. If the **Item** property cannot find the cell corresponding to the *Positions* argument, an error occurs.

The **Item** property is the default property for the **Cellset** object. The following syntax forms are interchangeable:

```
Cellset.Item ( Positions )  
Cellset ( Positions )
```

The *Positions* argument specifies which cell to return. You can specify the cell by ordinal position or by the position along each axis. When specifying the cell by position along each axis, you can specify the numeric value of the position or the names of the members for each position.

The ordinal position is a number that uniquely identifies one cell within the **Cellset**. Conceptually, cells are numbered in a **Cellset** as if the **Cellset** were a *p*-

dimensional array, where p is the number of axes. Cells are addressed in row-major order. Below is the formula for calculating the ordinal number of a cell:

If axis k has U_k members, the ordinal number of a cell whose tuple ordinals are $(S_0, S_1, S_2, \dots, S_{p-1})$ is

$$\sum_{i=0}^{p-1} S_i \times E_i \text{ where } E_0 = 1 \text{ and } E_i = \prod_{k=0}^{i-1} U_k$$

Σ represents the sum of the terms in the series and Π the product.

If member names are passed as strings to **Item**, the members must be listed in increasing order of the numeric axis identifiers. Within an axis, the members must be listed in increasing order of dimension nesting — that is, the outermost dimension's member comes first, followed by members of inner dimensions. Each dimension should be represented by a separate string, and the list of member strings should be separated by commas.

Note Retrieving cells by member name may not be supported by your [data provider](#). See the documentation for your provider for more information.

See Also

[Cell Object](#)

Applies To: [Cellset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

LevelDepth Property

Indicates the number of levels between the root of the [hierarchy](#) and a member.

Return Values

Returns a **Long** integer, and is read-only.

Remarks

Use the **LevelDepth** property to determine the distance of the [Member](#) object from the root level of the hierarchy. The **LevelDepth** of a member at the root level is 0. This corresponds to the [Depth](#) property of a [Level](#) object.

See Also

[Depth Property](#) | [Level Object](#)

Applies To: [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

LevelName Property

Indicates the name of the level of a member.

Return Values

Returns a **String** and is read-only.

Remarks

Use the **LevelName** property to retrieve the name of the level to which a member belongs. This corresponds to the [Name](#) property of a [Level](#) object.

See Also

[Level Object](#) | [Name Property](#)

Applies To: [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Name Property

Indicates the name of an object.

Return Values

Returns a **String** and is read-only.

Remarks

You can retrieve the **Name** property of an object by an ordinal reference, after which you can refer to the object directly by name. For example, if `cdf.CubeDefs(0).Name` yields "Bobs Video Store", you can refer to this [CubeDef](#) as `cdf.CubeDefs("Bobs Video Store")`.

See Also

[Visual Basic Example](#)

[Caption Property](#) | [Description Property](#) | [UniqueName Property](#)

Applies To: [Axis Object](#) | [Catalog Object](#) | [CubeDef Object](#) | [Dimension Object](#) | [Hierarchy Object](#) | [Level Object](#) | [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Ordinal Property (Cell)

Uniquely identifies a cell by its position within a cellset.

Return Values

Returns a **Long** integer and is read-only.

Remarks

The cell's ordinal value uniquely identifies the cell within a cellset. Conceptually, cells are numbered in a cellset as if the cellset were a p -dimensional array, where p is the number of [axes](#). Cells are numbered starting from zero in row-major order. Here is the formula for calculating the ordinal number of a cell:

If axis k has U_k members, the ordinal number of a cell whose tuple ordinals are $(S_0, S_1, S_2, \dots, S_{p-1})$ is

$$\sum_{i=0}^{p-1} S_i \times E_i \text{ where } E_0 = 1 \text{ and } E_i = \prod_{k=0}^{i-1} U_k$$

Σ represents the sum of the terms in the series and Π the product.

The cell's ordinal value can be used with the [Item](#) property of the [Cellset](#) object to quickly retrieve the [Cell](#).

See Also

[VBScript Example](#)

[Cellset Object](#) | [Item Property \(Cellset\)](#) | [Ordinal Property \(Position\)](#)

Applies To: [Cell Object](#)

ADO 2.5 MD API Reference

Ordinal Property (Position)

Uniquely identifies a position along an axis.

Return Values

Returns a **Long** integer and is read-only.

Remarks

The **Ordinal** of a [Position](#) object corresponds to the index of the **Position** within the [Positions](#) collection.

A cell can quickly be retrieved using the **Ordinal** of the **Position** along each axis with the [Item](#) property of the [Cellset](#) object.

See Also

[Cellset Object](#) | [Item Property \(Cellset\)](#) | [Ordinal Property \(Cell\)](#)

Applies To: [Position Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Parent Property

Indicates the member that is the [parent](#) of the current member in a [hierarchy](#).

Return Values

Returns a [Member](#) object and is read-only.

Remarks

A member that is at the top level of a hierarchy (the root) has no parent. This property is supported only on **Member** objects belonging to a [Level](#) object. An error occurs when this property is referenced from **Member** objects belonging to a [Position](#) object.

See Also

[Children Property](#)

Applies To: [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

ParentSameAsPrev Property

Indicates whether the [parent](#) of this position member is the same as the parent of the immediately preceding member.

Return Values

Returns a **Boolean** value and is read-only.

Remarks

This property is supported only on [Member](#) objects belonging to a [Position](#) object. An error occurs when this property is referenced from **Member** objects belonging to a [Level](#) object.

See Also

[DrilledDown Property](#)

Applies To: [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Source Property

Indicates the source for the data in the cellset.

Settings and Return Values

Sets or returns a **Variant**, and is read/write for closed [Cellset](#) objects and read-only for open **Cellset** objects. The **Variant** should contain a valid **String**, for example, an MDX query.

See Also

[Visual Basic Example](#)

[ActiveConnection Property](#) | [Open Method](#)

Applies To: [Cellset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

State Property

Indicates the current state of the cellset.

Return Values

Returns a **Long** integer indicating the current condition of the [Cellset](#) object and is read-only. The following values are valid: **adStateClosed** (0) and **adStateOpen** (1).

Remarks

To use the [ObjectStateEnum](#) constant names, you must have the ADO type library referenced in your project. See [Using ADO with ADO MD](#) for more information.

See Also

[Close Method](#) | [Open Method](#)

Applies To: [Cellset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Type Property

Indicates the type of the current member.

Return Values

Returns a [MemberTypeEnum](#) value and is read-only.

Remarks

This property is supported only on [Member](#) objects belonging to a [Level](#) object. An error occurs when this property is referenced from **Member** objects belonging to a [Position](#) object.

See Also

Applies To: [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

UniqueName Property

Indicates an unambiguous name for the current object.

Return Values

Returns a **String** and is read-only.

See Also

[Name Property](#)

Applies To: [Dimension Object](#) | [Hierarchy Object](#) | [Level Object](#) | [Member Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Value Property

Indicates the value of the current cell.

Return Values

Returns a **Variant** and is read-only.

See Also

[FormattedValue Property](#)

Applies To: [Cell Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

ADO MD Methods

Close	Closes an open cellset.
Open	Retrieves the results of a multidimensional query and returns the results to a cellset.

See Also

[ADO MD API Reference](#) | [ADO MD Code Examples](#) | [ADO MD Collections](#) | [ADO MD Enumerated Constants](#) | [ADO MD Object Model](#) | [ADO MD Objects](#) | [ADO MD Properties](#)

[© 1998-2002 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Close Method

Closes an open cellset.

Syntax

Cellset.Close

Remarks

Using the **Close** method to close a [Cellset](#) object will release the associated data, including data in any related [Cell](#), [Axis](#), [Position](#), or [Member](#) objects. Closing a **Cellset** does not remove it from memory; you can change its property settings and open it again later. To completely eliminate an object from memory, set the [object variable](#) to **Nothing**.

You can later call the [Open](#) method to reopen the **Cellset** using the same or another source string. While the **Cellset** object is closed, retrieving any properties or calling any methods that reference the underlying data or metadata generates an error.

See Also

[Axis Object](#) | [Cell Object](#) | [Member Object](#) | [Open Method](#) | [Position Object](#) | [State Property](#)

Applies To: [Cellset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

Open Method

Retrieves the results of a multidimensional query and returns the results to a cellset.

Syntax

Cellset.**Open** *Source*, *ActiveConnection*

Parameters

Source

Optional. A **VARIANT** that evaluates to a valid multidimensional query, such as a Multidimensional Expression (MDX) query. The *Source* argument corresponds to the [Source](#) property. For more information about MDX, see the OLE DB for OLAP documentation in the Microsoft Data Access Components SDK.

ActiveConnection

Optional. A **VARIANT** that evaluates to a string specifying either a valid ADO [Connection](#) object variable name or a definition for a connection. The *ActiveConnection* argument specifies the connection in which to open the [Cellset](#) object. If you pass a connection definition for this argument, ADO opens a new connection using the specified parameters. The *ActiveConnection* argument corresponds to the [ActiveConnection](#) property.

Remarks

The **Open** method generates an error if either of its parameters is omitted and its corresponding property value has not been set prior to attempting to open the **Cellset**.

See Also

[Visual Basic Example](#)

[ActiveConnection Property](#) | [Close Method](#) | [Connection Object](#) | [Source Property](#) | [State Property](#)

Applies To: [Cellset Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

ADO MD Enumerated Constants

To assist debugging, the ADO MD enumerated constants list a value for each constant. However, this value is purely advisory, and may change from one release of ADO MD to another. Your code should only depend on the name, not the actual value, of enumerated constants.

The following enumeration is defined.

- [MemberTypeEnum](#)

See Also

[ADO MD API Reference](#) | [ADO MD Code Examples](#) | [ADO MD Collections](#) | [ADO MD Methods](#) | [ADO MD Object Model](#) | [ADO MD Objects](#) | [ADO MD Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD API Reference

MemberTypeEnum

Specifies the setting for the [Type](#) property of a [Member](#) object.

Constant	Value	Description
adMemberAll	4	Indicates that the Member object represents all members of the level.
adMemberFormula	3	Indicates that the Member object is calculated using a formula expression.
adMemberMeasure	2	Indicates that the Member object belongs to the Measures dimension and represents a quantitative attribute.
adMemberRegular	1	Default. Indicates that the Member object represents an instance of a business entity.
adMemberUnknown	0	Cannot determine the type of the member.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

ADO MD Code Examples

Use the following code examples to learn how to use the ADO MD objects, methods, and properties. These examples are a subset of the sample applications installed with Microsoft SQL Server OLAP Services for SQL Server 7.0.

These examples use the MSOLAP OLE DB provider, and run against a Microsoft SQL Server OLAP Services local host. However, these examples are intended to show fundamental ADO MD programming techniques, and should be easily adapted to other data sources or providers.

- [ADO MD Code Examples in Visual Basic](#)
- [ADO MD Code Examples in Visual Basic, Scripting Edition](#)

See Also

[ADO MD API Reference](#) | [ADO MD Collections](#) | [ADO MD Enumerated Constants](#) | [ADO MD Methods](#) | [ADO MD Object Model](#) | [ADO MD Objects](#) | [ADO MD Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

ADO MD Code Examples in Visual Basic

The following examples require Microsoft Visual Basic version 5.0 with Service Pack 3 or Visual Basic version 6.0.

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

- [Cellset Example](#)
- [Catalog Example](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

Cellset Example (VB)

This Visual Basic project demonstrates the basics of using ADO MD to access cube data. It displays member captions for column and row headers, then displays formatted values of specific cells within the cellset.

```
Sub cmdCellSettoDebugWindow_Click()  
    On Error GoTo Error_cmdCellSettoDebugWindow_Click  
  
    Dim cat As New ADOXD.Catalog  
    Dim cst As New ADOXD.CellSet  
    Dim strServer As String  
    Dim strSource As String  
    Dim strColumnHeader As String  
    Dim strRowText As String  
    Dim i As Integer  
    Dim j As Integer  
    Dim k As Integer  
  
    Screen.MousePointer = vbHourglass  
  
    '*-----  
    '* Set Server to Local Host  
    '*-----  
    strServer = "localhost"  
  
    '*-----  
    '* Set MDX query string Source  
    '*-----  
    strSource = "SELECT {[Measures].members} ON COLUMNS," & _  
        "NON EMPTY [Store].[Store City].members ON ROWS FROM Sales"  
  
    '*-----  
    '* Set Active Connection  
    '*-----  
    cat.ActiveConnection = "Data Source=" & strServer & ";Provider=MSOLAP;  
        Catalog=Sales" & strServer  
  
    '*-----  
    '* Set Cell Set source to MDX query string  
    '*-----  
    cst.Source = strSource  
  
    '*-----  
    '* Set Cell Sets active connection to current connection  
    '*-----
```

```

Set cst.ActiveConnection = cat.ActiveConnection

'*-----
'* Open Cell Set
'*-----
cst.Open

'*-----
'* Allow space for Row Header Text
'*-----
strColumnHeader = vbTab & vbTab & vbTab & vbTab & vbTab & vbTab

'*-----
'* Loop through Column Headers
'*-----
For i = 0 To cst.Axes(0).Positions.Count - 1
    strColumnHeader = strColumnHeader & _
        cst.Axes(0).Positions(i).Members(0).Caption & vbTab & _
        vbTab & vbTab & vbTab
Next
Debug.Print vbTab & strColumnHeader & vbCrLf

'*-----
'* Loop through Row Headers and Provide data for each row
'*-----
strRowText = ""
For j = 0 To cst.Axes(1).Positions.Count - 1
    strRowText = strRowText & _
        cst.Axes(1).Positions(j).Members(0).Caption & vbTab & _
        vbTab & vbTab & vbTab
    For k = 0 To cst.Axes(0).Positions.Count - 1
        strRowText = strRowText & cst(k, j).FormattedValue & _
            vbTab & vbTab & vbTab & vbTab
    Next
    Debug.Print strRowText & vbCrLf
    strRowText = ""
Next

Screen.MousePointer = vbDefault

Exit Sub

Error_cmdCellSettoDebugWindow_Click:
Beep
Screen.MousePointer = vbDefault
Set cat = Nothing
Set cst = Nothing
MsgBox "The Following Error has occurred:" & vbCrLf & _
    Err.Description, vbCritical, " Error!"
Exit Sub

```

End Sub

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

Catalog Example (VB)

This Visual Basic project creates a new cube using MDX. Then, it documents the structure of a cube in a Microsoft Word document.

```
Sub cmdCreateDocForCube_Click()
    On Error GoTo Error_cmdCreateDocForCube_Click

    Dim cn As ADODB.Connection
    Dim s As String
    Dim strProvider As String
    Dim strDataSource As String
    Dim strSourceDSN As String
    Dim strSourceDSNSuffix As String
    Dim strCreateCube As String
    Dim strInsertInto As String

    '* -----
    '* The following code builds a cube file then documents the prop
    '* with an OLE Connection to Word 8.0
    '* -----

    '* -----
    '* Add Provider to the connection string.
    '* -----
    strProvider = "PROVIDER=MSOLAP"

    '* -----
    '* Add DataSource, the name of the file we will create.
    '* -----

    strDataSource = "DATA SOURCE=c:\DocumentCube.cub"

    '* -----
    '* Add Source DSN, the connection string for where the data come
    '* We need to quote the value so it is parsed as one value.
    '* This can either be an ODBC connection string or an OLE DB con
    '* string. (As returned by the Data Source Locator component.)
    '* -----
    strSourceDSN = "SOURCE_DSN=FoodMart 2000"

    '* -----
    '* Add CREATE CUBE. This defines the structure of the cube, but
    '* data in it. The BNF for is documented in the OLE DB for OLAP
    '* Programmer's Reference. Note that we can quote names with eit
```

'* double quotes or square brackets.

```
'* -----  
strCreateCube = "CREATECUBE=CREATE CUBE Sample( "  
strCreateCube = strCreateCube & "DIMENSION [Product],"  
strCreateCube = strCreateCube & "LEVEL [All Products] TYPE ALL,"  
strCreateCube = strCreateCube & "LEVEL [Product Family] ,"  
strCreateCube = strCreateCube & "LEVEL [Product Department] ,"  
strCreateCube = strCreateCube & "LEVEL [Product Category] ,"  
strCreateCube = strCreateCube & "LEVEL [Product Subcategory] ,"  
strCreateCube = strCreateCube & "LEVEL [Brand Name] ,"  
strCreateCube = strCreateCube & "LEVEL [Product Name] ,"  
strCreateCube = strCreateCube & "DIMENSION [Store],"  
strCreateCube = strCreateCube & "LEVEL [All Stores] TYPE ALL,"  
strCreateCube = strCreateCube & "LEVEL [Store Country] ,"  
strCreateCube = strCreateCube & "LEVEL [Store State] ,"  
strCreateCube = strCreateCube & "LEVEL [Store City] ,"  
strCreateCube = strCreateCube & "LEVEL [Store Name] ,"  
strCreateCube = strCreateCube & "DIMENSION [Store Type],"  
strCreateCube = strCreateCube & _  
    "LEVEL [All Store Type] TYPE ALL,"  
strCreateCube = strCreateCube & "LEVEL [Store Type] ,"  
strCreateCube = strCreateCube & "DIMENSION [Time] TYPE TIME,"  
strCreateCube = strCreateCube & "HIERARCHY [Column],"  
strCreateCube = strCreateCube & "LEVEL [All Time] TYPE ALL,"  
strCreateCube = strCreateCube & "LEVEL [Year] TYPE YEAR,"  
strCreateCube = strCreateCube & "LEVEL [Quarter] TYPE QUARTER,"  
strCreateCube = strCreateCube & "LEVEL [Month] TYPE MONTH,"  
strCreateCube = strCreateCube & "LEVEL [Week] TYPE WEEK,"  
strCreateCube = strCreateCube & "LEVEL [Day] TYPE DAY,"  
strCreateCube = strCreateCube & "HIERARCHY [Formula],"  
strCreateCube = strCreateCube & _  
    "LEVEL [All Formula Time] TYPE ALL,"  
strCreateCube = strCreateCube & "LEVEL [Year] TYPE YEAR,"  
strCreateCube = strCreateCube & "LEVEL [Quarter] TYPE QUARTER,"  
strCreateCube = strCreateCube & _  
    "LEVEL [Month] TYPE MONTH OPTIONS (SORTBYKEY) ,"  
strCreateCube = strCreateCube & "DIMENSION [Warehouse],"  
strCreateCube = strCreateCube & _  
    "LEVEL [All Warehouses] TYPE ALL,"  
strCreateCube = strCreateCube & "LEVEL [Country] ,"  
strCreateCube = strCreateCube & "LEVEL [State Province] ,"  
strCreateCube = strCreateCube & "LEVEL [City] ,"  
strCreateCube = strCreateCube & "LEVEL [Warehouse Name] ,"  
strCreateCube = strCreateCube & "MEASURE [Store Invoice] "  
strCreateCube = strCreateCube & "Function Sum "  
strCreateCube = strCreateCube & "Format '#.#'," "  
strCreateCube = strCreateCube & "MEASURE [Supply Time] "  
strCreateCube = strCreateCube & "Function Sum "  
strCreateCube = strCreateCube & "Format '#.#'," "  
strCreateCube = strCreateCube & "MEASURE [Warehouse Cost] "
```

```

strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#',"
strCreateCube = strCreateCube & "MEASURE [Warehouse Sales] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#',"
strCreateCube = strCreateCube & "MEASURE [Units Shipped] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#',"
strCreateCube = strCreateCube & "MEASURE [Units Ordered] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#')
' * -----
' * Add INSERT INTO. This defines where the data comes from, and
' * maps into the already-defined cube structure. Note that the
' * clause might just be passed through to the relational databas
' * So I could pass in a stored procedure, for example. If we ne
' * we could quote this whole thing. Note that the columns in th
' * can be in any order. One merely has to adjust the ordering o
' * list of level/measure names to match the SELECT ordering.
' * -----
strInsertInto = strInsertInto & _
    "INSERTINTO=INSERT INTO Sample( " & _
    "Product.[Product Family], Product.[Product Department],"
strInsertInto = strInsertInto & _
    "Product.[Product Category], Product.[Product Subcategory],"
strInsertInto = strInsertInto & _
    "Product.[Brand Name], Product.[Product Name],"
strInsertInto = strInsertInto & _
    "Store.[Store Country], Store.[Store State], Store.[Store Ci
strInsertInto = strInsertInto & _
    "Store.[Store Name], [Store Type].[Store Type], [Time].[Colu
strInsertInto = strInsertInto & _
    "[Time].Formula.Year, [Time].Formula.Quarter, " & _
    "[Time].Formula.Month.[Key],"
strInsertInto = strInsertInto & _
    "[Time].Formula.Month.Name, Warehouse.Country, " & _
    "Warehouse.[State Province],"
strInsertInto = strInsertInto & _
    "Warehouse.City, Warehouse.[Warehouse Name], " & _
    "Measures.[Store Invoice],"
strInsertInto = strInsertInto & _
    "Measures.[Supply Time], Measures.[Warehouse Cost], " & _
    "Measures.[Warehouse Sales],"
strInsertInto = strInsertInto & _
    "Measures.[Units Shipped], Measures.[Units Ordered] )"
' * -----
' * Add some options to the INSERT INTO if we need to. These can
' * if the SELECT clause is analyzed or just passed through, and
' * storage mode is MOLAP or ROLAP (DEFER_DATA).

```

```

'* strInsertInto = strInsertInto & " OPTIONS ATTEMPT_ANALYSIS"
'* -----

'* -----
'* Add the SELECT clause of the INSERT INTO statement. Note tha
'* merely concatenated onto the end of the INSERT INTO statement
'* Services will pass this through to the source database if una
'* parse it. Note that for OLAP Services to analyze the SELECT
'* each column must be qualified with the table name.
'* -----

strInsertInto = strInsertInto & _
    "SELECT product_class.product_family AS Col1,"
strInsertInto = strInsertInto & _
    "product_class.product_department AS Col2,"
strInsertInto = strInsertInto & "product_class.product_category
strInsertInto = strInsertInto & _
    "product_class.product_subcategory AS Col4,"
strInsertInto = strInsertInto & "product.brand_name AS Col5,"
strInsertInto = strInsertInto & "product.product_name AS Col6,"
strInsertInto = strInsertInto & "store.store_country AS Col7,"
strInsertInto = strInsertInto & "store.store_state AS Col8,"
strInsertInto = strInsertInto & "store.store_city AS Col9,"
strInsertInto = strInsertInto & "store.store_name AS Col10,"
strInsertInto = strInsertInto & "store.store_type AS Col11,"
strInsertInto = strInsertInto & "time_by_day.the_date AS Col12,"
strInsertInto = strInsertInto & "time_by_day.the_year AS Col13,"
strInsertInto = strInsertInto & "time_by_day.quarter AS Col14,"
strInsertInto = strInsertInto & "time_by_day.month_of_year AS Co
strInsertInto = strInsertInto & "time_by_day.the_month AS Col16,
strInsertInto = strInsertInto & "warehouse.warehouse_country AS
strInsertInto = strInsertInto & _
    "warehouse.warehouse_state_province AS Col18,"
strInsertInto = strInsertInto & "warehouse.warehouse_city AS Col
strInsertInto = strInsertInto & "warehouse.warehouse_name AS Col
strInsertInto = strInsertInto & _
    "inventory_fact_1997.store_invoice AS Col21,"
strInsertInto = strInsertInto & _
    "inventory_fact_1997.supply_time AS Col22,"
strInsertInto = strInsertInto & _
    "inventory_fact_1997.warehouse_cost AS Col23,"
strInsertInto = strInsertInto & _
    "inventory_fact_1997.warehouse_sales AS Col24,"
strInsertInto = strInsertInto & _
    "inventory_fact_1997.units_shipped AS Col25,"
strInsertInto = strInsertInto & _
    "inventory_fact_1997.units_ordered AS Col26 "
strInsertInto = strInsertInto & _
    "From [inventory_fact_1997], [product], [product_class], " &
    "[time_by_day], [store], [warehouse] "

```

```

strInsertInto = strInsertInto & _
    "Where [inventory_fact_1997].[product_id] = [product]." & _
    "[product_id] And "
strInsertInto = strInsertInto & _
    "[product].[product_class_id] = [product_class]." & _
    "[product_class_id] And "
strInsertInto = strInsertInto & _
    "[inventory_fact_1997].[time_id] = [time_by_day].[time_id] A
strInsertInto = strInsertInto & _
    "[inventory_fact_1997].[store_id] = [store].[store_id] And "
strInsertInto = strInsertInto & _
    "[inventory_fact_1997].[warehouse_id] = [warehouse].[warehou

'*-----
'* Create the cube by passing connection string to Open.
'*-----

Set cn = New ADODB.Connection
s = strProvider & ";" & strDataSource & ";" & strSourceDSN & ";"
    strCreateCube & ";" & strInsertInto & ";"
Screen.MousePointer = vbHourglass
cn.Open s

'*-----
'* Cube file is written to hard drive a Word Document can be pro
'* automating Word with VB
'*-----

Dim cat As New ADOMD.Catalog
Dim cdf As ADOMD.CubeDef
Dim i As Integer
Dim di As Integer
Dim hi As Integer
Dim le As Integer
Dim mem As Integer
Dim docWord As Word.Document
Dim rngCurrent As Word.Range
Dim SenCount As Integer
Dim strServer As String
Dim strSource As String
Dim strCubeName As String
Dim appWord As Object
'*-----
'* Connection is made to cube file
'*-----
cat.ActiveConnection = "DATA SOURCE=c:\DocumentCube.cub;Provider
'*-----
'* Cube Definition is set to Name of Cube in cube file

```

```

'*-----
Set cdf = cat.CubeDefs("Sample")

'*-----
'* Object is created to hold Word
'*-----
Set appWord = CreateObject("Word.Application")

'*-----
'* Create the document variable
'*-----
Set docWord = appWord.Documents.Add()

Set rngCurrent = docWord.Content

SenCount = 0

'*-----
'* Cube Title and Header written to Document
'*-----
With rngCurrent
    .InsertAfter "Report for Sample Cube"
    .InsertAfter vbCrLf
    SenCount = SenCount + 1
    docWord.Paragraphs(SenCount).Range.Bold = True
    docWord.Paragraphs(SenCount).Range.Underline = wdUnderlineSi
    docWord.Paragraphs(SenCount).Range.Italic = False
    docWord.Paragraphs(SenCount).Range.Font.Size = 18

    '*-----
    '* Properties of Cube are written to Document
    '*-----
    Debug.Print "Properties of Cube are written to Document"
    For i = 0 To cdf.Properties.Count - 1
        .InsertAfter "(" & i & ")" & cdf.Properties(i).Name & "
            cdf.Properties(i).Value
        .InsertAfter vbCrLf
        SenCount = SenCount + 1
        docWord.Paragraphs(SenCount).Range.Bold = False
        docWord.Paragraphs(SenCount).Range.Italic = True
        docWord.Paragraphs(SenCount).Range.Font.Size = 8
    Next i

    '*-----
    '* Dimension Name(s) written to Document
    '*-----
    Debug.Print "Dimension Name(s) written to Document"
    For di = 0 To cdf.Dimensions.Count - 1
        .InsertAfter "Dimension: " & cdf.Dimensions(di).Name
        .InsertAfter vbCrLf
    
```

```

SenCount = SenCount + 1
docWord.Paragraphs(SenCount).Range.Bold = True
docWord.Paragraphs(SenCount).Range.Italic = False
docWord.Paragraphs(SenCount).Range.Font.Size = 14

' * -----
' * Properties of Dimension are written to Document
' * -----
Debug.Print "Properties of Dimension are written to Docu
For i = 0 To cdf.Dimensions(di).Properties.Count - 1
    .InsertAfter "(" & i & ")" " & _
        cdf.Dimensions(di).Properties(i).Name & ": " & _
        cdf.Dimensions(di).Properties(i).Value
    .InsertAfter vbCrLf
SenCount = SenCount + 1
docWord.Paragraphs(SenCount).Range.Bold = False
docWord.Paragraphs(SenCount).Range.Italic = True
docWord.Paragraphs(SenCount).Range.Font.Size = 8
Next i

' * -----
' * Hierarchy Name(s) written to Document
' * -----
Debug.Print "Hierarchy Name(s) written to Document"
For hi = 0 To cdf.Dimensions(di).Hierarchies.Count - 1
    .InsertAfter vbTab & "Hierarchy: " & _
        cdf.Dimensions(di).Hierarchies(hi).Name
    .InsertAfter vbCrLf
SenCount = SenCount + 1
docWord.Paragraphs(SenCount).Range.Bold = True
docWord.Paragraphs(SenCount).Range.Italic = False
docWord.Paragraphs(SenCount).Range.Font.Size = 12

' * -----
' * Properties of Hierarchy are written to Document
' * -----
Debug.Print "Properties of Hierarchy are written to
For i = 0 To cdf.Dimensions(di).Hierarchies(hi).Prop
    .InsertAfter vbTab & "(" & i & ")" " & _
        cdf.Dimensions(di).Hierarchies(hi).Propertie
        ": " & cdf.Dimensions(di).Hierarchies(hi).Pr
    .InsertAfter vbCrLf
SenCount = SenCount + 1
docWord.Paragraphs(SenCount).Range.Bold = False
docWord.Paragraphs(SenCount).Range.Italic = True
docWord.Paragraphs(SenCount).Range.Font.Size = 8
Next i

' * -----

```

```

    '* Level Name(s) written to Document
    '*-----
    Debug.Print "Level Name(s) written to Document"
    For le = 0 To cdf.Dimensions(di).Hierarchies(hi).Levels
        .InsertAfter vbTab & vbTab & "Level: " & _
            cdf.Dimensions(di).Hierarchies(hi).Levels(le)
            " with a Member Count of: " & _
            cdf.Dimensions(di).Hierarchies(hi).Levels(le)
        .InsertAfter vbCrLf
        SenCount = SenCount + 1
        docWord.Paragraphs(SenCount).Range.Bold = True
        docWord.Paragraphs(SenCount).Range.Italic = False
        docWord.Paragraphs(SenCount).Range.Font.Size = 12

        '*-----
        '* Properties of Level are written to Document
        '*-----
        Debug.Print "Properties of Level are written to"
        For i = 0 To _
            cdf.Dimensions(di).Hierarchies(hi).Levels(le)
            .InsertAfter vbTab & vbTab & "(" & i & ")" "
                cdf.Dimensions(di).Hierarchies(hi).Levels(le)
                cdf.Dimensions(di).Hierarchies(hi).Levels(le)
            .InsertAfter vbCrLf
            SenCount = SenCount + 1
            docWord.Paragraphs(SenCount).Range.Bold = False
            docWord.Paragraphs(SenCount).Range.Italic = False
            docWord.Paragraphs(SenCount).Range.Font.Size = 12
        Next i

    Next le
    Next hi
Next di

    '*-----
    '* Set Word Document to visible
    '*-----
    Debug.Print "Set Word Document to visible"
    appWord.Visible = True
End With

Screen.MousePointer = vbDefault

    '*-----
    '* Set Word Object to nothing to drop OLE connection
    '*-----
    Set appWord = Nothing
Exit Sub

Error_cmdCreateDocForCube_Click:

```

```
cn.Cancel  
Set cn = Nothing  
Screen.MousePointer = vbDefault  
MsgBox Err.Description  
End Sub
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

ADO MD Code Examples in Visual Basic, Scripting Edition

The following examples require Microsoft Active Server Pages (ASP) for Microsoft Internet Information Server 4.0.

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

- [Axis Example](#)
- [Members Example](#)
- [CubeDef Example](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

Axis Example (VBScript)

This Active Server Page displays OLAP data from an MDX Query string and writes the resulting cellset to an HTML table structure.

```
<%@ Language=VBScript %>
<%
!*****
!*** Active Server Page displays OLAP data from default
!*** MDX Query string and writes resulting cell set to HTML table
!*** structure.
!*****
Response.Buffer=True
Response.Expires=0
%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY bgcolor=Ivory>
<FONT FACE=Verdana>

<%

Dim cat,cst,i,j,strSource,csw,intDC0,intDC1,intPC0,intPC1

!*****
!*** Set Connection Objects for Multidimensional Catalog and Cellset
!*****
Set cat = Server.CreateObject("ADOMD.Catalog")
Set cst = Server.CreateObject("ADOMD.CellSet")

!*****
!*** Use default settings of a known OLAP Server
!*** for Server Name for Connection Set Server Name Session Object
!*** to default value
!*****
!*** Must set OLAPServerName to OLAP Server that is
!*** present on network
!*****
    OLAPServerName = "Please set to present OLAP Server"
    cat.ActiveConnection = "Data Source='" & OLAPServerName & _
        "';Initial Catalog='FoodMart';Provider='msolap';"

!*****
```

```

'*** Use default MDX Query string of a known query that works
'*** with default server Set MDXQuery Session Object to default valu
'*****
    strSource = strSource & "SELECT "
    strSource = strSource & "{[Measures].members} ON COLUMNS,"
    strSource = strSource & _
        "NON EMPTY [Store].[Store City].members ON ROWS"
    strSource = strSource & " FROM Sales"

'*****
'*** Set Cell Set Source property to strSource to be passed on cell
'*****
    cst.Source = strSource

'*****
'*** Set Cell Sets Active connection to use the current Catalogs Act
'*** connection
'*****
Set cst.ActiveConnection = cat.ActiveConnection

'*****
'*** Using Open method, Open cell set
'*****
cst.Open

'*****
'*** Set Dimension Counts minus 1 for Both Axes to intDC0, intDC1
'*** Set Position Counts minus 1 for Both Axes to intPC0, intPC1
'*****
intDC0 = cst.Axes(0).DimensionCount-1
intDC1 = cst.Axes(1).DimensionCount-1

intPC0 = cst.Axes(0).Positions.Count - 1
intPC1 = cst.Axes(1).Positions.Count - 1

'*****
'*** Create HTML Table structure to hold MDX Query return Record set
'*****
    Response.Write "<Table width=100% border=1>"

'*****
'*** Loop to create Column header
'*****
    For h=0 to intDC0
        Response.Write "<TR>"

'*****
'*** Loop to create spaces in front of Column headers
'*** to align with Row header
'*****

```

```

        For c=0 to intDC1
            Response.Write "<TD></TD>"
        Next

'*****
'*** Iterate through Axes(0) Positions writing member captions to ta
'*** header
'*****
        For i = 0 To intPC0
            Response.Write "<TH>"
            Response.Write "<FONT size=-2>"
            Response.Write cst.Axes(0).Positions(i).Members(h).Capti
            Response.Write "</FONT>"
            Response.Write "</TH>"
        Next
        Response.Write "</TR>"
    Next
'*****
'*** Use Array values for row header formatting to provide
'*** spaces under beginning row header titles
'*****
        For j = 0 To intPC1
            Response.Write "<TR>"
            For h=0 to intDC1
                Response.Write "<TD><B>"
                Response.Write "<FONT size=-2>"
                Response.Write cst.Axes(1).Positions(j).Members(h).Capti
                Response.Write "</FONT>"
                Response.Write "</B></TD>"
            Next
            For k = 0 To intPC0
                Response.Write "<TD align=right bgcolor="
                Response.Write csw
                Response.Write ">"
                Response.Write "<FONT size=-2>"
                Response.Write cst(k, j).FormattedValue
                Response.Write "</FONT>"
                Response.Write "</TD>"
            Next
            Response.Write "</TR>"
        Next
        Response.Write "</Table>"

%>
</FONT>
</BODY>
</HTML>

```

ADO 2.5 MD 

Members Example (VBScript)

This sample uses an MDX query string to retrieve OLAP data and writes the resulting cellset to an HTML table structure using column spanning features for multiple-dimension cellsets.

```
<%@ Language=VBScript %>
<%
!*****
!*** Active Server Page displays OLAP data from default or provided
!*** MDX Query string and writes resulting cell set to HTML table
!*** structure. This ASP provides colspan features for multiple
!*** dimension cell sets.
!*****
Response.Buffer=True
Response.Expires=0
%>
<html>
<head>
</head>
<body bgcolor="Ivory">
<font FACE="Verdana">

<%

Dim cat,cst,i,j,strSource,csw,LevelValue,intDC0,intDC1,intPC0, intPC
!*****
!*** Gather Server Name and MDX Query Strings from text box and
!*** text area and assign them to Session Objects of same name
!*****
Session("ServerName")=Request.Form("strServerName")
Session("InitialCatalog")=Request.Form("strInitialCatalog")
Session("MDXQuery")=Request.Form("MDXQuery")

!*****
!*** Set Connection Objects for Multi dimensional Catalog and Cell S
!*****
Set cat = Server.CreateObject("ADOMD.Catalog")
Set cst = Server.CreateObject("ADOMD.CellSet")

!*****
!*** Check to see if the Session Object Server Name is present
!*** If present then: Create Active Connection using Server Name
!*** and MSOLAP as connection Provider
!*** If not present then: Use default settings of a known OLAP Serve
```

```

'*** for Server Name for Connection Set Server Name Session Object
'*** to default value
'*****
If Len(Session("ServerName")) > 0 Then
    cat.ActiveConnection = "Data Source='" & Session("ServerName") &
        "';Initial Catalog='" & Session("InitialCatalog") & _
        "';Provider='msolap';"
Else

'*****
'*** Must set OLAPServerName to OLAP Server that is
'*** present on network
'*****
    OLAPServerName = "Please set to present OLAP Server"
    cat.ActiveConnection = "Data Source=" & OLAPServerName & _
        ";Initial Catalog=FoodMart;Provider=msolap;"
    Session("ServerName") = OLAPServerName
    Session("InitialCatalog") = "FoodMart"
End if

'*****
'*** Check to see if the Session Object MDXQuery is present
'*** If present then: Set strSource using MDXQuery Session Object
'*** If not present then: Use default MDX Query string of a known qu
'*** that works with default server Set MDXQuery Session Object to
'*** default value
'*****
If Len(Session("MDXQuery")) < 5 Then
    strSource = strSource & "SELECT "
    strSource = strSource & "CROSSJOIN({[Store].[Store Country].MEMBE
    strSource = strSource & "{[Measures].[Store " & _
        "Invoice],[Measures].[Supply Time]}) ON COLUMNS,"
    strSource = strSource & "CROSSJOIN({[Time].[Year].MEMBERS},"
    strSource = strSource & "CROSSJOIN({[Store Type].[Store " & _
        "Type].Members},{[Product].[Product Family].members})) ON ROWS
    strSource = strSource & " FROM Warehouse"
Else
    strSource = Session("MDXQuery")
End if

'*****
'*** Set Cell Set Source property to strSource to be passed on cell
'*** open method
'*****
    cst.Source = strSource

'*****
'*** Set Cell Sets Active connection to use the current Catalogs Act
'*** connection
'*****
Set cst.ActiveConnection = cat.ActiveConnection

```

```

!*****
!*** Using Open method, Open cell set
!*****
cst.Open

!*****
!*** Standard HTML to collect Server Name and MDX Query Information
!*** Note that post action posts back to same page to process
!*** thus using state of Session Variables to change look of page
!*****
%>
<form action="ASPADOComplex.asp" method="POST" id="form1" name="form"
<table>
<tr><td align="left">
<b>Olap Server name:</b><br><input type="text" id="strServerName" na
<br>
<b>Catalog name:</b><br><input type="text" id="strInitialCatalog" na
</td><td align="center">
<b>MDX Query:</b><br>
<textarea rows="7" cols="70" id="textareaMDX" name="MDXQuery" wrap="
<%=Session("MDXQuery")%>
</textarea>
</td></tr>
</table>
<table>
<tr><td>
<input type="submit" value="Submit MDX Query" id="submit1" name="sub
</td><td>
<input type="reset" value="Reset" id="reset1" name="reset1">
</td></tr>
</table>
</form>
<p align="left">
<font color="Black" size="-3">
<%=strSource%>
</font>
</p>
<%
!*****
!*** Set Dimension Counts minus 1 for Both Axes to intDC0, intDC1
!*** Set Position Counts minus 1 for Both Axes to intPC0, intPC1
!*****
intDC0 = cst.Axes(0).DimensionCount-1
intDC1 = cst.Axes(1).DimensionCount-1

intPC0 = cst.Axes(0).Positions.Count - 1
intPC1 = cst.Axes(1).Positions.Count - 1

```

```

'*****
'*** Create HTML Table structure to hold MDX Query return Record set
'*****
Response.Write "<Table width=100% border=1>"

'*****
'*** Loop to create Column header for all Dimensions based
'*** on Count of Dimensions for Axes(0)
'*****
For h=0 to intDC0
    Response.Write "<TR>"

'*****
'*** Loop to create spaces in front of Column headers
'*** to align with Row headers
'*****
    For c=0 to intDC1
        Response.Write "<TD></TD>"
    Next

'*****
'*** Check current dimension to see if equal to Last Dimension
'*** If True: Write Table header titles normally to HTML output with
'*** ColSpan value
'*** If False: Write Table header titles with ColSpan values to HTML
'*** output
'*****
    If h = intDC0 then

'*****
'*** Iterate through Axes(0) Positions writing member captions to ta
'*** header
'*****
        For i = 0 To intPC0
            Response.Write "<TH>"
            Response.Write "<FONT size=-2>"
            Response.Write cst.Axes(0).Positions(i).Members(h).Caption
            Response.Write "</FONT>"
            Response.Write "</TH>"
        Next
    Else

'*****
'*** Iterate through Axes(0) Positions writing member captions to ta
'*** header taking into account for the span of columns for duplicat
'*** member captions
'*****
        CaptionCount = 1
        LastCaption = cst.Axes(0).Positions(0).Members(h).Caption
        Response.Write "<TH"

```

```

        For t=1 to intPC0

'*****
'*** Check to see if LastCaption is equal to current members caption
'*** If True: Add one to CaptionCount to increase Colspan value
'*** If False: Write Table header titles with ColSpan values to HTML
'*** output using current CaptionCount for Colspan and LastCaption f
'*** header string
'*****
        If LastCaption = _
            cst.Axes(0).Positions(t).Members(h).Caption then
                CaptionCount = CaptionCount+1

'*****
'*** Check if at last position
'*** If True: Write HTML to finish table row using current
'*** CaptionCount and LastCaption
'*****
                If t = intPC0 then
                    Response.Write " colspan=" & CaptionCount & _
                        "><FONT size=-2>" & LastCaption & "</FONT></TH>"
                End if

                Else
                    Response.Write " colspan=" & CaptionCount & _
                        "><FONT size=-2>" & LastCaption & "</FONT></TH><TH"
                    CaptionCount = 1
                    LastCaption=cst.Axes(0).Positions(t).Members(h).Caption
                End if
            Next
        End if
        Response.Write "</TR>"
    Next

'*****
'*** Iterate through Axes(1) Positions first writing member captions
'*** to table row headers then writing cell set data to table struct
'*****
        Dim aryRows()
        Dim intArray,Marker
        intArray=0

'*****
'*** Set value of Array for row header formatting
'*****
        For a=1 To intDC1
            intArray = intArray+(intPC1+1)
        Next
        intArray = intArray-1

```

```

ReDim aryRows(intArray)
Marker=0

!*****
!*** Use Array values for row header formatting to provide
!*** spaces under beginning row header titles
!*****

For j = 0 To intPC1
    Response.Write "<TR>"
    For h=0 to intDC1
        If h=intDC1 then
            Response.Write "<TD><B>"
            Response.Write "<FONT size=-2>"
            Response.Write cst.Axes(1).Positions(j).Members(h).Ca
            Response.Write "</FONT>"
            Response.Write "</B></TD>"
        Else
            aryRows(Marker) = _
                cst.Axes(1).Positions(j).Members(h).Caption
            If Marker < intDC1 then
                Response.Write "<TD><B>"
                Response.Write "<FONT size=-2>"
                Response.Write _
                    cst.Axes(1).Positions(j).Members(h).Caption
                Response.Write "</FONT>"
                Response.Write "</B></TD>"
                Marker = Marker + 1
            Else
                If aryRows(Marker) = aryRows(Marker - intDC1) then
                    Response.Write "<TD>&nbsp;</TD>"
                    Marker = Marker + 1
                Else
                    Response.Write "<TD><B>"
                    Response.Write "<FONT size=-2>"
                    Response.Write _
                        cst.Axes(1).Positions(j).Members(h).Caption
                    Response.Write "</FONT>"
                    Response.Write "</B></TD>"
                    Marker = Marker + 1
                End if
            End if
        End if
    Next
End if

!*****
!*** Alternates Cell background color
!*****

If (j+1) Mod 2 = 0 Then
    csw = "#cccccc"
Else

```

```

        csw = "#ccffff"
    End If
    For k = 0 To intPC0
        Response.Write "<TD align=right bgcolor="
        Response.Write csw
        Response.Write ">"
        Response.Write "<FONT size=-2>"

!*****
!*** FormattedValue property pulls data
!*****

        Response.Write cst(k, j).FormattedValue
        Response.Write "</FONT>"
        Response.Write "</TD>"
    Next
    Response.Write "</TR>"
Next
Response.Write "</Table>"

%>
</font>
</body>
</html>

```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 MD 

CubeDef Example (VBScript)

This example displays cube metadata on a web page.

```
<%@ Language=VBScript %>
<%
Response.Buffer=True
'Response.Expires=0
%>
<html>
<head>
<meta NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</head>
<body>

<%
Server.ScriptTimeout=360
Dim cat,cdf,di,hi,le,mem,strServer,strSource,strCubeName

!*****
!*** Set Session Variables
!*****
Session("CubeName") = Request.Form("strCubeName")
Session("CatalogName") = Request.Form("strCatalogName")
Session("ServerName") = Request.Form("strServerName")
Session("chkDim") = Request.Form("chkDimension")
Session("chkHier") = Request.Form("chkHierarchy")
Session("chkLev") = Request.Form("chkLevel")

!*****
!*** Create Catalog Object
!*****
Set cat = Server.CreateObject("ADOMD.Catalog")

If Len(Session("ServerName")) > 0 Then
    cat.ActiveConnection = "Data Source='" & Session("ServerName") &
Else
!*****
!*** Must set OLAPServerName to OLAP Server that is
!*** present on network
!*****
OLAPServerName = "Please set to present OLAP Server"
    cat.ActiveConnection = "Data Source=" & OLAPServerName & _
        ";Initial Catalog=FoodMart;Provider=msolap;"
    Session("ServerName") = OLAPServerName
```

```

    Session("InitialCatalog") = "FoodMart"
End if

If Len(Session("CubeName")) > 0 Then
    Set cdf = cat.CubeDefs(Session("CubeName"))
Else
    Set cdf = cat.CubeDefs("Sales")
    Session("CubeName")="Sales"
End if

!*****
!*** Collect Information in HTML Form
!*****
%>
<form action="ASPADOCubeDoc.asp" method="post" id="form1" name="form"
<table>
    <tr>
        <td>
            <b>Olap Server name: </b><br><input type="text" id="strServer
            <b>Catalog Name: </b><br><input type="text" id="strCatalogNar
            <b>Cube Name: </b><br><input type="text" id="strCubeName" nar
        </td>
        <td <TD>
            <b>Add Property Detail: </b><br>
            Dimension Detail: <input type="checkbox" id="chkDimension"
            Hierarchy Detail: <input type="checkbox" id="chkHierarchy"
            Level Detail: <input type="checkbox" id="chkLevel" name="ch
        </td>
    </tr>
</table>
<input type="submit" value="Cube Information" id="submit1" name="sub
</form>
<%

!*****
!*** Start of Report
!*****
Response.Write "<H3>Report for " & Session("CubeName") & " Cube</H3>
Response.Write "<OL TYPE='i'>"

!*****
!*** Show properties of Cube
!*****
    For i = 0 To cdf.Properties.Count - 1
        Response.Write "<LI>"
        Response.Write "<FONT size=-2>" & cdf.Properties(i).N

```



```

Response.Write "<FONT size=2><B>Level: " & _
    cdf.Dimensions(di).Hierarchies(hi).Levels(le).Name
    " with a Member Count of: " & _
    cdf.Dimensions(di).Hierarchies(hi).Levels(le)._
    Properties("LEVEL_CARDINALITY") & "</B></FONT>"
If Request.Form("chkLevel") = "on" Then
Response.Write "<OL TYPE='1'>"
For i = 0 To
    cdf.Dimensions(di).Hierarchies(hi).Levels(le)._
    Properties.Count - 1
Response.Write "<LI>"
Response.Write "<FONT size=-2>" & _
    cdf.Dimensions(di).Hierarchies(hi).Levels(le)
    Properties(i).Name & ": " & _
    cdf.Dimensions(di).Hierarchies(hi).Levels(le)
    Properties(i).Value & "</FONT>"
Next
Response.Write "</OL>"
End If
Next
Response.Write "</UL>"
Next
Response.Write "</UL>"
Next
Response.Write "</UL>"
%>
</body>
</html>

```

ADOX 2.5 API Reference

ADOX API Reference

This section of the ADOX documentation contains topics for each ADOX object, collection, method, and property, as well as example code when appropriate. For more information, search for a specific topic in the index or refer to the following topics:

- [ADOX Objects](#)
- [ADOX Collections](#)
- [ADOX Properties](#)
- [ADOX Methods](#)
- [ADOX Enumerated Constants](#)
- [ADOX Code Examples](#)

See Also

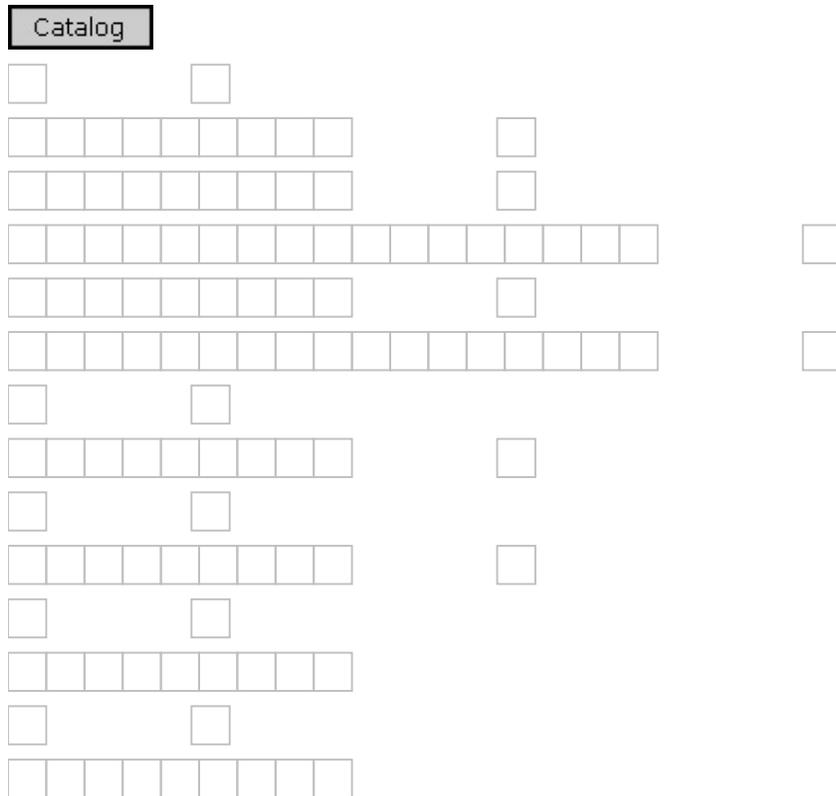
[ADO API Reference](#) | [ADO MD API Reference](#) | [ADOX Code Examples](#) | [ADOX Collections](#) | [ADOX Enumerated Constants](#) | [ADOX Methods](#) | [ADOX Object Model](#) | [ADOX Objects](#) | [ADOX Properties](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

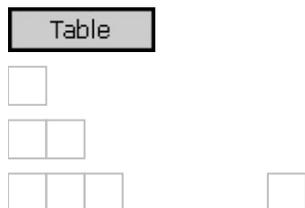
ADOX 2.5 API Reference

ADOX Object Model

The following diagram illustrates how objects are represented and related in ADOX. For more information about a specific object or collection, see the specific reference topic, or [ADOX Objects](#) and [ADOX Collections](#).



Each of the [Table](#), [Index](#), and [Column](#) objects also has a standard ADO [Properties](#) collection.



See Also

[ADOX API Reference](#) | [ADOX Code Examples](#) | [ADOX Collections](#) | [ADOX Enumerated Constants](#) | [ADOX Methods](#) | [ADOX Objects](#) | [ADOX Properties](#) | [Microsoft ADOX Programmer's Reference](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ADOX Objects

ADOX Object Summary

Object	Description
Catalog	Contains collections that describe the schema catalog of a data source.
Column	Represents a column from a table, index, or key.
Group	Represents a group account that has access permissions within a secured database.
Index	Represents an index from a database table.
Key	Represents a primary, foreign, or unique key field from a database table.
Procedure	Represents a stored procedure .
Table	Represents a database table, including columns, indexes, and keys.
User	Represents a user account that has access permissions within a secured database.
View	Represents a filtered set of records or a virtual table.

The relationships between these objects are illustrated in the [ADOX Object Model](#).

Each object can be contained in its corresponding collection. For example, a **Table** object can be contained in a [Tables](#) collection. For more information, see [ADOX Collections](#) or a specific collection topic.

See Also

[ADOX API Reference](#) | [ADOX Collections](#) | [ADOX Object Model](#) | [Microsoft ADOX Programmer's Reference](#)

ADOX 2.5 API Reference

Catalog Object

Contains collections ([Tables](#), [Views](#), [Users](#), [Groups](#), and [Procedures](#)) that describe the schema catalog of a data source.

Catalog

Remarks

You can modify the **Catalog** object by adding or removing objects or by modifying existing objects. Some providers may not support all of the **Catalog** objects or may support only viewing schema information.

With the properties and methods of a **Catalog** object, you can:

- Open the catalog by setting the [ActiveConnection](#) property to an ADO [Connection](#) object or a valid connection string.
- Create a new catalog with the [Create](#) method.
- Determine the owners of the objects in a **Catalog** with the [GetObjectOwner](#) and [SetObjectOwner](#) methods.

See Also

[Catalog ActiveConnection Property Example \(VB\)](#) | [Command and CommandText Properties Example \(VB\)](#) | [Connection Close Method, Table Type Property Example \(VB\)](#) | [Create Method Example \(VB\)](#) | [Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#) | [Parameters Collection, Command Property Example \(VB\)](#) | [ParentCatalog Property Example \(VB\)](#) | [Procedures Append Method Example \(VB\)](#) | [Procedures Delete Method Example \(VB\)](#) | [Procedures Refresh Method Example](#)

[\(VB\)](#) | [Views and Fields Collections Example \(VB\)](#) | [Views Append Method Example \(VB\)](#) | [Views Collection, CommandText Property Example \(VB\)](#) | [Views Delete Method Example \(VB\)](#) | [Views Refresh Method Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Groups Collection](#) | [Procedures Collection](#) | [Tables Collection](#) | [Users Collection](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Catalog Object Properties, Methods, and Events

Properties/Collections

[ActiveConnection Property](#)

[Groups Collection](#)

[Procedures Collection](#)

[Tables Collection](#)

[Users Collection](#)

[Views Collection](#)

Methods

[Create Method](#)

[GetObjectOwner Method](#)

[SetObjectOwner Method](#)

Events

None.

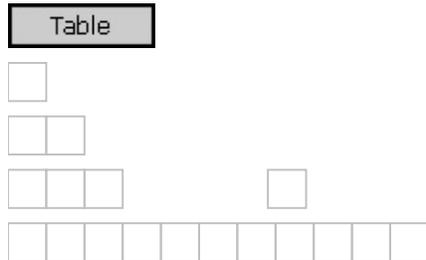
See Also

Applies To: [Catalog Object](#)

ADOX 2.5 API Reference

Column Object

Represents a column from a table, index, or key.



Remarks

The following code creates a new **Column**:

```
Dim obj As New Column
```

With the properties and collections of a **Column** object, you can:

- Identify the column with the [Name](#) property.
- Specify the data type of the column with the [Type](#) property.
- Determine if the column is fixed-length, or if it can contain null values with the [Attributes](#) property.
- Specify the maximum size of the column with the [DefinedSize](#) property.
- For numeric data values, specify the scale with the [NumericScale](#) property.
- For numeric data value, specify the maximum precision with the [Precision](#) property.
- Specify the [Catalog](#) that owns the column with the [ParentCatalog](#) property.
- For key columns, specify the name of the related column in the related table with the [RelatedColumn](#) property.
- For index columns, specify whether the sort order is ascending or descending with the [SortOrder](#) property.
- Access [provider](#)-specific properties with the [Properties](#) collection.

Note Not all properties of **Column** objects may be supported by your data provider. An error will occur if you have set a value for a property that the provider does not support. For new **Column** objects, the error will occur

when the object is appended to the collection. For existing objects, the error will occur when setting the property.

When creating **Column** objects, the existence of an appropriate default value for an optional property does not guarantee that your provider supports the property. For more information about which properties your provider supports, see your provider documentation.

See Also

[Columns and Tables Append Methods, Name Property Example \(VB\) | Connection Close Method, Table Type Property Example \(VB\) | Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\) | NumericScale and Precision Properties Example \(VB\) | ParentCatalog Property Example \(VB\) | SortOrder Property Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Columns Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Column Object Properties, Methods, and Events

Properties/Collections

[Attributes Property](#)

[DefinedSize Property](#)

[Name Property](#)

[NumericScale Property](#)

[ParentCatalog Property](#)

[Precision Property](#)

[Properties Collection](#)

[RelatedColumn Property](#)

[SortOrder Property](#)

[Type Property \(Column\)](#)

Methods

None.

Events

None.

See Also

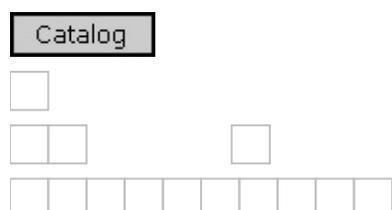
Applies To: [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Group Object

Represents a group account that has access permissions within a secured database.



Remarks

The [Groups](#) collection of a [Catalog](#) represents all the catalog's group accounts. The **Groups** collection for a [User](#) represents only the group to which the user belongs.

With the properties, collections, and methods of a **Group** object, you can:

- Identify the group with the [Name](#) property.
- Determine whether a group has read, write, or delete permissions with the [GetPermissions](#) and [SetPermissions](#) methods.
- Access the user accounts that have memberships in the group with the [Users](#) collection.
- Access [provider](#)-specific properties with the [Properties](#) collection.

See Also

[Properties, Methods, and Events](#) | [Groups Collection](#) | [Users Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Group Object Properties, Methods, and Events

Properties/Collections

[Name Property](#)

[Properties Collection](#)

[Users Collection](#)

Methods

[GetPermissions Method](#)

[SetPermissions Method](#)

Events

None.

See Also

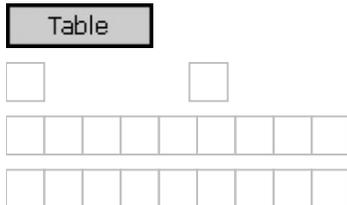
Applies To: [Group Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Index Object

Represents an index from a database table.



Remarks

The following code creates a new **Index**:

```
Dim obj As New Index
```

With the properties and collections of an **Index** object, you can:

- Identify the index with the [Name](#) property.
- Access the database columns of the index with the [Columns](#) collection.
- Specify whether the index keys must be unique with the [Unique](#) property.
- Specify whether the index is the primary key for a table with the [PrimaryKey](#) property.
- Specify whether records that have null values in their index fields have index entries with the [IndexNulls](#) property.
- Specify whether the index is clustered with the [Clustered](#) property.
- Access [provider](#)-specific index properties with the [Properties](#) collection.

Notes An error will occur when appending a [Column](#) to the **Columns** collection of an **Index** if the **Column** does not exist in a [Table](#) object already appended to the [Tables](#) collection.

Your data provider may not support all properties of **Index** objects. An error will occur if you have set a value for a property that is not supported by the provider. For new **Index** objects, the error will occur when the object is appended to the collection. For existing objects, the error will occur when setting the property.

When creating **Index** objects, the existence of an appropriate default value for an optional property does not guarantee that your provider supports the property. For more information about which properties your provider supports, see your provider documentation.

See Also

[Indexes Append Method Example \(VB\)](#) | [IndexNulls Property Example \(VB\)](#) | [PrimaryKey and Unique Properties Example \(VB\)](#) | [SortOrder Property Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Columns Collection](#) | [Indexes Collection](#) | [Properties Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Index Object Properties, Methods, and Events

Properties/Collections

[Clustered Property](#)

[Columns Collection](#)

[IndexNulls Property](#)

[Name Property](#)

[PrimaryKey Property](#)

[Properties Collection](#)

[Unique Property](#)

Methods

None.

Events

None.

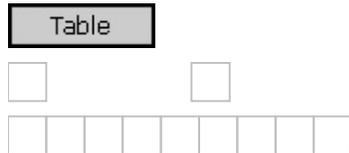
See Also

Applies To: [Index Object](#)

ADOX 2.5 API Reference

Key Object

Represents a primary, foreign, or unique key field from a database table.



Remarks

The following code creates a new **Key**:

```
Dim obj As New Key
```

With the properties and collections of a **Key** object, you can:

- Identify the key with the [Name](#) property.
- Determine whether the key is primary, foreign, or unique with the [Type](#) property.
- Access the database columns of the key with the [Columns](#) collection.
- Specify the name of the related table with the [RelatedTable](#) property.
- Determine the action performed on deletion or update of a primary key with the [DeleteRule](#) and [UpdateRule](#) properties.

See Also

[Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Columns Collection](#) | [Keys Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Key Object Properties, Methods, and Events

Properties/Collections

[Columns Collection](#)

[DeleteRule Property](#)

[Name Property](#)

[RelatedTable Property](#)

[Type Property \(Key\)](#)

[UpdateRule Property](#)

Methods

None.

Events

None.

See Also

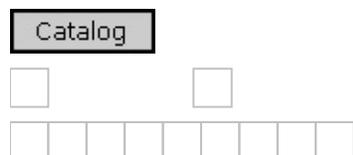
Applies To: [Key Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Procedure Object

Represents a [stored procedure](#). When used in conjunction with the ADO [Command](#) object, the **Procedure** object can be used for adding, deleting, or modifying stored procedures.



Remarks

The **Procedure** object allows you to create a stored procedure without having to know or use the provider's "CREATE PROCEDURE" syntax.

With the properties of a **Procedure** object, you can:

- Identify the procedure with the [Name](#) property.
- Specify the ADO **Command** object that can be used to create or execute the procedure with the [Command](#) property.
- Return date information with the [DateCreated](#) and [DateModified](#) properties.

See Also

[Command and CommandText Properties Example \(VB\)](#) | [Parameters Collection, Command Property Example \(VB\)](#) | [Procedures Append Method Example \(VB\)](#) | [Procedures Delete Method Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Procedures Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Procedure Object Properties, Methods, and Events

Properties

[Command Property](#)

[DateCreated Property](#)

[DateModified Property](#)

[Name Property](#)

Methods

None.

Events

None.

See Also

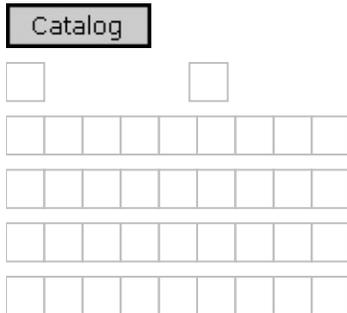
Applies To: [Procedure Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Table Object

Represents a database table including columns, indexes, and keys.



Remarks

The following code creates a new **Table**:

```
Dim obj As New Table
```

With the properties and collections of a **Table** object, you can:

- Identify the table with the [Name](#) property.
- Determine the type of table with the [Type](#) property.
- Access the database columns of the table with the [Columns](#) collection.
- Access the indexes of the table with the [Indexes](#) collection.
- Access the keys of the table with the [Keys](#) collection.
- Specify the [Catalog](#) that owns the table with the [ParentCatalog](#) property.
- Return date information with the [DateCreated](#) and [DateModified](#) properties.
- Access [provider](#)-specific table properties with the [Properties](#) collection.

Note Your data provider may not support all properties of **Table** objects. An error will occur if you have set a value for a property that the provider does not support. For new **Table** objects, the error will occur when the object is appended to the collection. For existing objects, the error will occur when setting the property.

When creating **Table** objects, the existence of an appropriate default value for an optional property does not guarantee that your provider supports the

property. For more information about which properties your provider supports, see your provider documentation.

See Also

[Catalog ActiveConnection Property Example \(VB\)](#) | [Columns and Tables Append Methods, Name Property Example \(VB\)](#) | [Connection Close Method, Table Type Property Example \(VB\)](#) | [Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#) | [ParentCatalog Property Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Columns Collection](#) | [Indexes Collection](#) | [Keys Collection](#) | [Properties Collection](#) | [Tables Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Table Object Properties, Methods, and Events

Properties/Collections

[DateCreated Property](#)

[DateModified Property](#)

[Indexes Collection](#)

[Keys Collection](#)

[Name Property](#)

[ParentCatalog Property](#)

[Type Property \(Table\)](#)

Methods

None.

Events

None.

See Also

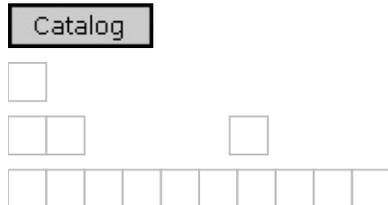
Applies To: [Table Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

User Object

Represents a user account that has access permissions within a secured database.



Remarks

The [Users](#) collection of a [Catalog](#) represents all the catalog's users. The **Users** collection for a [Group](#) represents only the users of the specific group.

With the properties, collections, and methods of a **User** object, you can:

- Identify the user with the [Name](#) property.
- Change the password for a user with the [ChangePassword](#) method.
- Determine whether a user has read, write, or delete permissions with the [GetPermissions](#) and [SetPermissions](#) methods.
- Access the groups to which a user belongs with the [Groups](#) collection.
- Access [provider](#)-specific properties with the [Properties](#) collection.

See Also

[GetPermissions and SetPermissions Methods Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Groups Collection](#) | [Users Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

User Object Properties, Methods, and Events

Properties/Collections

[Groups Collection](#)

[Name Property](#)

[Properties Collection](#)

Methods

[ChangePassword Method](#)

[GetPermissions Method](#)

[SetPermissions Method](#)

Events

None.

See Also

Applies To: [User Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

View Object

Represents a filtered set of records or a virtual table. When used in conjunction with the ADO [Command](#) object, the **View** object can be used for adding, deleting, or modifying views.



Remarks

A view is a virtual table, created from other database tables or views. The **View** object allows you to create a view without having to know or use the provider's "CREATE VIEW" syntax.

With the properties of a **View** object, you can:

- Identify the view with the [Name](#) property.
- Specify the ADO **Command** object that can be used to add, delete, or modify views with the [Command](#) property.
- Return date information with the [DateCreated](#) and [DateModified](#) properties.

See Also

[Views and Fields Collections Example \(VB\)](#) | [Views Append Method Example \(VB\)](#) | [Views Collection, CommandText Property Example \(VB\)](#) | [Views Delete Method Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Views Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

View Object Properties, Methods, and Events

Properties

[Command Property](#)

[DateCreated Property](#)

[DateModified Property](#)

[Name Property](#)

Methods

None.

Events

None.

See Also

Applies To: [View Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ADOX Collections

Columns	Contains all Column objects of a table, index, or key.
Groups	Contains all stored Group objects of a catalog or user.
Indexes	Contains all Index objects of a table.
Keys	Contains all Key objects of a table.
Procedures	Contains all Procedure objects of a catalog.
Tables	Contains all Table objects of a catalog.
Users	Contains all stored User objects of a catalog or group.
Views	Contains all View objects of a catalog.

See Also

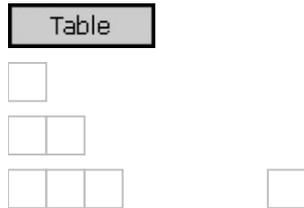
[ADOX API Reference](#) | [ADOX Object Model](#) | [ADOX Objects](#) | [Microsoft ADOX Programmer's Reference](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Columns Collection

Contains all [Column](#) objects of a table, index, or key.



Remarks

The [Append](#) method for a **Columns** collection is unique for ADOX. You can:

- Add a new column to the collection with the **Append** method.

The remaining properties and methods are standard to ADO collections. You can:

- Access a column in the collection with the [Item](#) property.
- Return the number of columns contained in the collection with the [Count](#) property.
- Remove a column from the collection with the [Delete](#) method.
- Update the objects in the collection to reflect the current database's schema with the [Refresh](#) method.

Note An error will occur when appending a **Column** to the **Columns** collection of an [Index](#) if the **Column** does not exist in a [Table](#) that is already appended to the [Tables](#) collection.

See Also

[Columns and Tables Append Methods, Name Property Example \(VB\) | Connection Close Method, Table Type Property Example \(VB\) | Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\) | ParentCatalog Property Example \(VB\) | SortOrder Property](#)

[Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Columns Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method \(Columns\)](#)

[Delete Method \(Collections\)](#)

[Refresh Method](#)

Events

None.

See Also

Applies To: [Columns Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Groups Collection

Contains all stored [Group](#) objects of a catalog or user.



Remarks

The **Groups** collection of a [Catalog](#) represents all of the catalog's group accounts. The **Groups** collection for a [User](#) represents only the group to which the user belongs.

The [Append](#) method for a **Groups** collection is unique for ADOX. You can:

- Add a new security group to the collection with the **Append** method.

The remaining properties and methods are standard to ADO collections. You can:

- Access a group in the collection with the [Item](#) property.
- Return the number of groups contained in the collection with the [Count](#) property.
- Remove a group from the collection with the [Delete](#) method.
- Update the objects in the collection to reflect the current database's schema with the [Refresh](#) method.

Note Before appending a **Group** object to the **Groups** collection of a **User** object, a **Group** object with the same [Name](#) as the one to be appended must already exist in the **Groups** collection of the **Catalog**.

See Also

[Properties, Methods, and Events](#) | [Catalog Object](#) | [Group Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Groups Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method \(Groups\)](#)

[Delete Method \(Collections\)](#)

[Refresh Method](#)

Events

None.

See Also

Applies To: [Groups Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Indexes Collection

Contains all [Index](#) objects of a table.



Remarks

The [Append](#) method for an **Indexes** collection is unique for ADOX. You can:

- Add a new index to the collection with the **Append** method.

The remaining properties and methods are standard to ADO collections. You can:

- Access an index in the collection with the [Item](#) property.
- Return the number of indexes contained in the collection with the [Count](#) property.
- Remove an index from the collection with the [Delete](#) method.
- Update the objects in the collection to reflect the current database's schema with the [Refresh](#) method.

See Also

[Indexes Append Method Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Index Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Indexes Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method \(Indexes\)](#)

[Delete Method \(Collections\)](#)

[Refresh Method](#)

Events

None.

See Also

Applies To: [Indexes Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Keys Collection

Contains all [Key](#) objects of a table.

Table



Remarks

The [Append](#) method for a **Keys** collection is unique for ADOX. You can:

- Add a new key to the collection with the **Append** method.

The remaining properties and methods are standard to ADO collections. You can:

- Access a key in the collection with the [Item](#) property.
- Return the number of keys contained in the collection with the [Count](#) property.
- Remove a key from the collection with the [Delete](#) method.
- Update the objects in the collection to reflect the current database's schema with the [Refresh](#) method.

See Also

[Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Key Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Keys Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method \(Keys\)](#)

[Delete Method \(Collections\)](#)

[Refresh Method](#)

Events

None.

See Also

Applies To: [Keys Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Procedures Collection

Contains all [Procedure](#) objects of a catalog.



Remarks

The [Append](#) method for a **Procedures** collection is unique for ADOX. You can:

- Add a new procedure to the collection with the **Append** method.

The remaining properties and methods are standard to ADO collections. You can:

- Access a procedure in the collection with the [Item](#) property.
- Return the number of procedures contained in the collection with the [Count](#) property.
- Remove a procedure from the collection with the [Delete](#) method.
- Update the objects in the collection to reflect the current database's schema with the [Refresh](#) method.

See Also

[Command and CommandText Properties Example \(VB\)](#) | [Parameters Collection, Command Property Example \(VB\)](#) | [Procedures Append Method Example \(VB\)](#) | [Procedures Delete Method Example \(VB\)](#) | [Procedures Refresh Method Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Catalog Object](#) | [Procedure Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Procedures Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method \(Procedures\)](#)

[Delete Method \(Collections\)](#)

[Refresh Method](#)

Events

None.

See Also

Applies To: [Procedures Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Tables Collection

Contains all [Table](#) objects of a catalog.



Remarks

The [Append](#) method for a **Tables** collection is unique for ADOX. You can:

- Add a new table to the collection with the **Append** method.

The remaining properties and methods are standard to ADO collections. You can:

- Access a table in the collection with the [Item](#) property.
- Return the number of tables contained in the collection with the [Count](#) property.
- Remove a table from the collection with the [Delete](#) method.
- Update the objects in the collection to reflect the current database's schema with the [Refresh](#) method.

Some providers may return other schema objects, such as a View, in the Tables collection. Therefore, some ADOX collections may contain references to the same object. Should you delete the object from one collection, the change will not be visible in another collection that references the deleted object until the Refresh method is called on the collection. For example, with the OLE DB Provider for Microsoft Jet, Views are returned with the Tables collection. If you drop a View, you must Refresh the Tables collection before the collection will reflect the change.

See Also

[Catalog ActiveConnection Property Example \(VB\)](#) | [Columns and Tables Append Methods, Name Property Example \(VB\)](#) | [Connection Close Method,](#)

[Table Type Property Example \(VB\)](#) | [Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Catalog Object](#) | [Table Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Tables Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method \(Tables\)](#)

[Delete Method \(Collections\)](#)

[Refresh Method](#)

Events

None.

See Also

Applies To: [Tables Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Users Collection

Contains all stored [User](#) objects of a catalog or group.

Catalog



Remarks

The **Users** collection of a [Catalog](#) represents all the catalog's users. The **Users** collection for a [Group](#) represents only the users that have a membership in the specific group.

The [Append](#) method for a **Users** collection is unique for ADOX. You can:

- Add a new user to the collection with the **Append** method.

The remaining properties and methods are standard to ADO collections. You can:

- Access a user in the collection with the [Item](#) property.
- Return the number of users contained in the collection with the [Count](#) property.
- Remove a user from the collection with the [Delete](#) method.
- Update the objects in the collection to reflect the current database's schema with the [Refresh](#) method.

Note Before appending a **User** object to the **Users** collection of a **Group** object, a **User** object with the same [Name](#) as the one to be appended must already exist in the **Users** collection of the **Catalog**.

See Also

[GetPermissions and SetPermissions Methods Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Catalog Object](#) | [User Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Users Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method \(Users\)](#)

[Delete Method \(Collections\)](#)

[Refresh Method](#)

Events

None.

See Also

Applies To: [Users Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Views Collection

Contains all [View](#) objects of a catalog.



Remarks

The [Append](#) method for a **Views** collection is unique for ADOX. You can:

- Add a new view to the collection with the **Append** method.

The remaining properties and methods are standard to ADO collections. You can:

- Access a view in the collection with the [Item](#) property.
- Return the number of views contained in the collection with the [Count](#) property.
- Remove a view from the collection with the [Delete](#) method.
- Update the objects in the collection to reflect the current database's schema with the [Refresh](#) method.

See Also

[Views and Fields Collections Example \(VB\)](#) | [Views Append Method Example \(VB\)](#) | [Views Collection, CommandText Property Example \(VB\)](#) | [Views Delete Method Example \(VB\)](#) | [Views Refresh Method Example \(VB\)](#)

[Properties, Methods, and Events](#) | [Catalog Object](#) | [View Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Views Collection Properties, Methods, and Events

Properties

[Count Property](#)

[Item Property](#)

Methods

[Append Method \(Views\)](#)

[Delete Method \(Collections\)](#)

[Refresh Method](#)

Events

None.

See Also

Applies To: [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ADOX Properties

ActiveConnection	Indicates the ADO Connection object to which the catalog belongs.
Attributes	Describes column characteristics.
Clustered	Indicates whether the index is clustered.
Command	Specifies an ADO Command object that can be used to create or execute the procedure.
Count	Indicates the number of objects in a collection.
DateCreated	Indicates the date the object was created.
DateModified	Indicates the date the object was last modified.
DefinedSize	Indicates the stated maximum size of the column.
DeleteRule	Indicates the action performed when a primary key is deleted.
IndexNulls	Indicates whether records that have null values in their index fields have index entries.
Item	Indicates a specific member of a collection, by name or ordinal number.
Name	Indicates the name of the object.
NumericScale	Indicates the scale of a numeric value in the column.
ParentCatalog	Specifies the parent catalog of a table or column to provide access to provider-specific properties.
Precision	Indicates the maximum precision of data values in the column.
PrimaryKey	Indicates whether the index represents the primary key on the table.
RelatedColumn	Indicates the name of the related column in the related table (key columns only).
RelatedTable	Indicates the name of the related table.
SortOrder	Indicates the sort sequence for the column (index columns only).
Type (Column)	Indicates the data type of a column.
Type (Key)	Indicates the data type of the Key.

Type (Table)	Indicates the type of a table.
Unique	Indicates whether the index keys must be unique.
UpdateRule	Indicates the action performed when a primary key is updated.

See Also

[ADOX API Reference](#) | [Microsoft ADOX Programmer's Reference](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ActiveConnection Property

Indicates the ADO [Connection](#) object to which the [Catalog](#) belongs.

Settings and Return Values

Sets a **Connection** object or a **String** containing the definition for a connection. Returns the active **Connection** object.

Remarks

The default value is a null object reference.

See Also

[Catalog ActiveConnection Property Example \(VB\)](#) | [Command and CommandText Properties Example \(VB\)](#) | [Connection Close Method, Table Type Property Example \(VB\)](#) | [Parameters Collection, Command Property Example \(VB\)](#) | [Procedures Append Method Example \(VB\)](#) | [Procedures Delete Method Example \(VB\)](#) | [Procedures Refresh Method Example \(VB\)](#) | [Views and Fields Collections Example \(VB\)](#) | [Views Append Method Example \(VB\)](#) | [Views Collection, CommandText Property Example \(VB\)](#) | [Views Refresh Method Example \(VB\)](#)

[Create Method](#)

Applies To: [Catalog Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Attributes Property

Describes column characteristics.

Settings and Return Values

Sets or returns a **Long** value. The value specifies characteristics of the table represented by the [Column](#) object and can be a combination of [ColumnAttributesEnum](#) constants. The default value is zero (0), which is neither **adColFixed** nor **adColNullable**.

See Also

[Attributes Property Example \(VB\)](#)

Applies To: [Column Object](#) | [Property Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Clustered Property

Indicates whether the index is clustered.

Settings and Return Values

Sets and returns a **Boolean** value.

Remarks

The default value is **False**.

This property is read-only on [Index](#) objects already appended to a collection.

See Also

[Clustered Property Example \(VB\)](#)

Applies To: [Index Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Command Property

Specifies an ADO [Command](#) object that can be used to create or execute the procedure.

Settings and Return Values

Sets or returns a valid ADO **Command** object.

Remarks

An error will occur when getting and setting this property if the [provider](#) does not support persisting commands.

See Also

[Command and CommandText Properties Example \(VB\)](#) | [Parameters Collection, Command Property Example \(VB\)](#) | [Views and Fields Collections Example \(VB\)](#) | [Views Collection, CommandText Property Example \(VB\)](#)

Applies To: [Procedure Object](#) | [View Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

DateCreated Property

Indicates the date the object was created.

Return Values

Returns a **VARIANT** value specifying the date created. The value is null if **DateCreated** is not supported by the [provider](#).

Remarks

The **DateCreated** property is null for newly appended objects. After appending a new [View](#) or [Procedure](#), you must call the [Refresh](#) method of the [Views](#) or [Procedures](#) collection to obtain values for the **DateCreated** property.

See Also

[DateCreated and DateModified Properties Example \(VB\)](#)

[DateModified Property](#)

Applies To: [Procedure Object](#) | [Table Object](#) | [View Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

DateModified Property

Indicates the date the object was last modified.

Return Values

Returns a **VARIANT** value specifying the date modified. The value is null if **DateModified** is not supported by the [provider](#).

Remarks

The **DateModified** property is null for newly appended objects. After appending a new [View](#) or [Procedure](#), you must call the [Refresh](#) method of the [Views](#) or [Procedures](#) collection to obtain values for the **DateModified** property.

See Also

[DateCreated and DateModified Properties Example \(VB\)](#)

[DateCreated Property](#)

Applies To: [Procedure Object](#) | [Table Object](#) | [View Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

DefinedSize Property

Indicates the stated maximum size of the column.

Settings and Return Values

Sets and returns a **Long** value that is the maximum length in characters of data values.

Remarks

The default value is zero (0).

This property is read-only for [Column](#) objects already appended to a collection.

See Also

[DefinedSize Property Example \(VB\)](#)

Applies To: [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

DeleteRule Property

Indicates the action performed when a primary key is deleted.

Settings and Return Values

Sets and returns a **Long** value that can be one of the [RuleEnum](#) constants. The default value is **adRINone**.

Remarks

This property is read-only on [Key](#) objects already appended to a collection.

See Also

[DeleteRule Property Example \(VB\)](#)

Applies To: [Key Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

IndexNulls Property

Indicates whether records that have null values in their index fields have index entries.

Settings and Return Values

Sets and returns an [AllowNullsEnum](#) value. The default value is **adIndexNullsDisallow**.

Remarks

This property is read-only on [Index](#) objects already appended to a collection.

See Also

[IndexNulls Property Example \(VB\)](#)

Applies To: [Index Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Name Property

Indicates the name of the object.

Settings and Return Values

Sets or returns a **String** value.

Remarks

Names do not have to be unique within a collection.

The **Name** property is read/write on [Column](#), [Group](#), [Key](#), [Index](#), [Table](#), and [User](#) objects. The **Name** property is read-only on [Catalog](#), [Procedure](#), and [View](#) objects.

For read/write objects (**Column**, **Group**, **Key**, **Index**, **Table** and **User** objects), the default value is an empty string ("").

Notes For keys, this property is read-only on **Key** objects already appended to a collection.

For tables, this property is read-only for **Table** objects already appended to a collection.

See Also

[Columns and Tables Append Methods, Name Property Example \(VB\)](#) | [Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#) | [ParentCatalog Property Example \(VB\)](#)

Applies To: [Column Object](#) | [Group Object](#) | [Index Object](#) | [Key Object](#) | [Procedure Object](#) | [Property Object](#) | [Table Object](#) | [User Object](#) | [View Object](#)

ADOX 2.5 API Reference

NumericScale Property

Indicates the scale of a numeric value in the column.

Settings and Return Values

Sets and returns a **Byte** value that is the scale of data values in the column when the [Type](#) property is **adNumeric** or **adDecimal**. **NumericScale** is ignored for all other data types.

Remarks

The default value is zero (0).

NumericScale is read-only for [Column](#) objects already appended to a collection.

See Also

[NumericScale and Precision Properties Example \(VB\)](#)

[Type Property \(Column\)](#)

Applies To: [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ParentCatalog Property

Specifies the parent catalog of a table or column to provide access to provider-specific properties.

Settings and Return Values

Sets and returns a [Catalog](#) object. Setting **ParentCatalog** to an open **Catalog** allows access to provider-specific properties prior to appending a table or column to a **Catalog** collection.

Remarks

Some [data providers](#) allow provider-specific property values to be written only at creation (when a table or column is appended to its **Catalog** collection). To access these properties before appending these objects to a **Catalog**, specify the **Catalog** in the **ParentCatalog** property first.

An error occurs when the table or column is appended to a different **Catalog** than the **ParentCatalog**.

See Also

[ParentCatalog Property Example \(VB\)](#)

Applies To: [Column Object](#) | [Table Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Precision Property

Indicates the maximum precision of data values in the column.

Settings and Return Values

Sets and returns a **Long** value that is the maximum precision of data values in the column when the [Type](#) property is a numeric type. **Precision** is ignored for all other data types.

Remarks

The default value is zero (0).

This property is read-only for [Column](#) objects already appended to a collection.

See Also

[NumericScale and Precision Properties Example \(VB\)](#)

[Type Property \(Column\)](#)

Applies To: [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

PrimaryKey Property

Indicates whether the index represents the primary key on the table.

Settings and Return Values

Sets and returns a **Boolean** value.

Remarks

The default value is **False**.

This property is read-only on [Index](#) objects already appended to a collection.

See Also

[PrimaryKey and Unique Properties Example \(VB\)](#)

Applies To: [Index Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

RelatedColumn Property

Indicates the name of the related column in the related table (key columns only).

Settings and Return Values

Sets and returns a **String** value that is the name of the related column in the related table.

Remarks

The default value is an empty string ("").

This property is read-only for [Column](#) objects already appended to a collection.

See Also

[Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#)

[Key Object](#)

Applies To: [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

RelatedTable Property

Indicates the name of the related table.

Settings and Return Values

Sets and returns a **String** value.

Remarks

The default value is an empty string ("").

If the key is a foreign key, then **RelatedTable** is the name of the table that contains the key.

See Also

[Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#)

Applies To: [Key Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

SortOrder Property

Indicates the sort sequence for the column (index columns only).

Settings and Return Values

Sets and returns a **Long** value that can be one of the [SortOrderEnum](#) constants. The default value is **adSortAscending**.

Remarks

This property only applies to [Column](#) objects in the [Columns](#) collection of an [Index](#).

See Also

[SortOrder Property Example \(VB\)](#)

[Columns Collection](#) | [Index Object](#)

Applies To: [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Type Property (Column)

Indicates the data type of a column.

Settings and Return Values

Sets or returns a **Long** value that can be one of the [DataTypeEnum](#) constants. The default value is **adVarChar**.

Remarks

This property is read/write until the [Column](#) object is appended to a collection or to another object, after which it is read-only.

See Also

[ParentCatalog Property Example \(VB\)](#)

[Type Property \(Key\)](#) | [Type Property \(Table\)](#)

Applies To: [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Type Property (Key)

Indicates the type of the key.

Settings and Return Values

Sets or returns a **Long** value that can be one of the [KeyTypeEnum](#) constants. The default value is **adKeyPrimary**.

Remarks

This property is read-only on [Key](#) objects already appended to a collection.

See Also

[Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#)

[Type Property \(Column\)](#) | [Type Property \(Table\)](#)

Applies To: [Key Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Type Property (Table)

Indicates the type of a table.

Return Values

Returns a **String** value that specifies the type of table; for example, "TABLE", "SYSTEM TABLE", or "GLOBAL TEMPORARY".

Remarks

This property is read-only.

See Also

[Catalog ActiveConnection Property Example \(VB\)](#) | [Connection Close Method, Table Type Property Example \(VB\)](#)

[Type Property \(Column\)](#) | [Type Property \(Key\)](#)

Applies To: [Table Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Unique Property

Indicates whether the index keys must be unique.

Settings and Return Values

Sets and returns a **Boolean** value.

Remarks

The default value is **False**.

This property is read-only on [Index](#) objects already appended to a collection.

See Also

[PrimaryKey and Unique Properties Example \(VB\)](#)

[Key Object](#)

Applies To: [Index Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

UpdateRule Property

Indicates the action performed when a primary key is updated.

Settings and Return Values

Sets and returns a **Long** value that can be one of the [RuleEnum](#) constants. The default value is **adRINone**.

Remarks

This property is read-only on [Key](#) objects already appended to the collection.

See Also

[Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#)

Applies To: [Key Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ADOX Methods

Append (Columns)	Adds a new Column object to the Columns collection.
Append (Groups)	Adds a new Group object to the Groups collection.
Append (Indexes)	Adds a new Index object to the Indexes collection.
Append (Keys)	Adds a new Key object to the Keys collection.
Append (Procedures)	Adds a new Procedure object to the Procedures collection.
Append (Tables)	Adds a new Table object to the Tables collection.
Append (Users)	Adds a new User object to the Users collection.
Append (Views)	Adds a new View object to the Views collection.
ChangePassword	Changes the password for a user account.
Create	Creates a new catalog.
Delete	Removes an object from a collection.
GetObjectOwner	Returns the owner of an object in a catalog.
GetPermissions	Returns the permissions for a group or user on an object.
SetObjectOwner	Specifies the owner of an object in a catalog.
SetPermissions	Specifies the permissions for group or user on an object.

See Also

[ADOX API Reference](#) | [Microsoft ADOX Programmer's Reference](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Append Method (Columns)

Adds a new [Column](#) object to the [Columns](#) collection.

Syntax

```
Columns.Append Column [, Type] [, DefinedSize]
```

Parameters

Column

The **Column** object to append or the name of the column to create and append.

Type

Optional. A **Long** value that specifies the data type of the column. The *Type* parameter corresponds to the [Type](#) property of a **Column** object.

DefinedSize

Optional. A **Long** value that specifies the size of the column. The *DefinedSize* parameter corresponds to the [DefinedSize](#) property of a **Column** object.

Note An error will occur when appending a **Column** to the **Columns** collection of an [Index](#) if the **Column** does not exist in a [Table](#) that is already appended to the [Tables](#) collection.

See Also

[Columns and Tables Append Methods, Name Property Example \(VB\)](#) | [Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#) | [ParentCatalog Property Example \(VB\)](#)

[Append Method \(Groups\)](#) | [Append Method \(Indexes\)](#) | [Append Method \(Keys\)](#) | [Append Method \(Procedures\)](#) | [Append Method \(Tables\)](#) | [Append Method \(Users\)](#) | [Append Method \(Views\)](#)

Applies To: [Columns Collection](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

Append Method (Groups)

Adds a new [Group](#) object to the [Groups](#) collection.

Syntax

```
Groups.Append Group
```

Parameters

Group

The **Group** object to append or the name of the group to create and append.

Remarks

The **Groups** collection of a [Catalog](#) represents all of the catalog's group accounts. The **Groups** collection for a [User](#) represents only the group to which the user belongs.

An error will occur if the provider does not support creating groups.

Note Before appending a **Group** object to the **Groups** collection of a **User** object, a **Group** object with the same [Name](#) as the one to be appended must already exist in the **Groups** collection of the **Catalog**.

See Also

[Groups and Users Append, ChangePassword Methods Example \(VB\)](#)

[Append Method \(Columns\)](#) | [Append Method \(Indexes\)](#) | [Append Method \(Keys\)](#) | [Append Method \(Procedures\)](#) | [Append Method \(Tables\)](#) | [Append Method \(Users\)](#) | [Append Method \(Views\)](#)

Applies To: [Groups Collection](#)

ADOX 2.5 API Reference

Append Method (Indexes)

Adds a new [Index](#) object to the [Indexes](#) collection.

Syntax

```
Indexes.Append Index [, Columns]
```

Parameters

Index

The **Index** object to append or the name of the index to create and append.

Columns

Optional. A **Variant** value that specifies the name(s) of the column(s) to be indexed. The *Columns* parameter corresponds to the value(s) of the [Name](#) property of a [Column](#) object or objects.

Remarks

The *Columns* parameter can take either the name of a column or an array of column names.

An error will occur if the provider does not support creating indexes.

See Also

[Indexes Append Method Example \(VB\)](#)

[Append Method \(Columns\)](#) | [Append Method \(Groups\)](#) | [Append Method \(Keys\)](#) | [Append Method \(Procedures\)](#) | [Append Method \(Tables\)](#) | [Append Method \(Users\)](#) | [Append Method \(Views\)](#)

Applies To: [Indexes Collection](#)

ADOX 2.5 API Reference

Append Method (Keys)

Adds a new [Key](#) object to the [Keys](#) collection.

Syntax

```
Keys.Append Key [, KeyType] [, Column] [, RelatedTable] [, RelatedCo
```

Parameters

Key

The **Key** object to append or the name of the key to create and append.

KeyType

Optional. A **Long** value that specifies the type of key. The *Key* parameter corresponds to the [Type](#) property of a **Key** object.

Column

Optional. A **String** value that specifies the name of the column to be indexed. The *Columns* parameter corresponds to the value of the [Name](#) property of a [Column](#) object.

RelatedTable

Optional. A **String** value that specifies the name of the related table. The *RelatedTable* parameter corresponds to the value of the **Name** property of a [Table](#) object.

RelatedColumn

Optional. A **String** value that specifies the name of the related column for a foreign key. The *RelatedColumn* parameter corresponds to the value of the **Name** property of a **Column** object.

Remarks

The *Columns* parameter can take either the name of a column or an array of column names.

See Also

[Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#)

[Append Method \(Columns\)](#) | [Append Method \(Groups\)](#) | [Append Method \(Indexes\)](#) | [Append Method \(Procedures\)](#) | [Append Method \(Tables\)](#) | [Append Method \(Users\)](#) | [Append Method \(Views\)](#)

Applies To: [Keys Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Append Method (Procedures)

Adds a new [Procedure](#) object to the [Procedures](#) collection.

Syntax

Procedures.Append *Name*, *Command*

Parameters

Name

A **String** value that specifies the name of the procedure to create and append.

Command

An ADO [Command](#) object that represents the procedure to create and append.

Remarks

Creates a new procedure in the data source with the name and attributes specified in the **Command** object.

If the command text that the user specifies represents a view rather than a procedure, the behavior is dependent upon the provider being used. **Append** will fail if the provider does not support persisting commands.

Note When using the OLE DB Provider for Microsoft Jet, the **Procedures** collection **Append** method will allow you to specify a **View** rather than a **Procedure** in the *Command* parameter. The **View** will be added to the data source and will be added to the **Procedures** collection. After the **Append**, if the **Procedures** and **Views** collections are refreshed, the **View** will no longer be in the **Procedures** collection and will appear in the **Views** collection.

See Also

[Procedures Append Method Example \(VB\)](#)

[Append Method \(Columns\)](#) | [Append Method \(Groups\)](#) | [Append Method \(Indexes\)](#) | [Append Method \(Keys\)](#) | [Append Method \(Tables\)](#) | [Append Method \(Users\)](#) | [Append Method \(Views\)](#)

Applies To: [Procedures Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Append Method (Tables)

Adds a new [Table](#) object to the [Tables](#) collection.

Syntax

```
Tables.Append Table
```

Parameters

Table

A **VARIANT** value that contains a reference to the **Table** to append or the name of the table to create and append.

Remarks

An error will occur if the [provider](#) does not support creating tables.

See Also

[Columns and Tables Append Methods, Name Property Example \(VB\)](#) | [ParentCatalog Property Example \(VB\)](#)

[Append Method \(Columns\)](#) | [Append Method \(Groups\)](#) | [Append Method \(Indexes\)](#) | [Append Method \(Keys\)](#) | [Append Method \(Procedures\)](#) | [Append Method \(Users\)](#) | [Append Method \(Views\)](#)

Applies To: [Tables Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Append Method (Users)

Adds a new [User](#) object to the [Users](#) collection.

Syntax

```
Users.Append User[, Password]
```

Parameters

User

A **Variant** value that contains the **User** object to append or the name of the user to create and append.

Password

Optional. A **String** value that contains the password for the user. The *Password* parameter corresponds to the value specified by the [ChangePassword](#) method of a **User** object.

Remarks

The **Users** collection of a [Catalog](#) represents all the catalog's users. The **Users** collection for a [Group](#) represents only the users that have a membership in the specific group.

An error will occur if the provider does not support creating users.

Note Before appending a **User** object to the **Users** collection of a **Group** object, a **User** object with the same [Name](#) as the one to be appended must already exist in the **Users** collection of the **Catalog**.

See Also

[Groups and Users Append, ChangePassword Methods Example \(VB\)](#)

[Append Method \(Columns\)](#) | [Append Method \(Groups\)](#) | [Append Method \(Indexes\)](#) | [Append Method \(Keys\)](#) | [Append Method \(Procedures\)](#) | [Append](#)

[Method \(Tables\)](#) | [Append Method \(Views\)](#)

Applies To: [Users Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Append Method (Views)

Creates a new [View](#) object and appends it to the [Views](#) collection.

Syntax

```
Views.Append Name, Command
```

Parameters

Name

A **String** value that specifies the name of the view to create.

Command

An ADO [Command](#) object that represents the view to create.

Remarks

Creates a new view in the data source with the name and attributes specified in the **Command** object.

If the command text that the user specifies represents a procedure rather than a view, the behavior is dependent upon the provider. **Append** will fail if the provider does not support persisting commands.

Note When using the OLE DB Provider for Microsoft Jet, the **Views** collection **Append** method will allow you to specify a **Procedure** rather than a **View** in the *Command* parameter. The **Procedure** will be added to the data source and will be added to the **Views** collection. After the **Append**, if the **Procedures** and **Views** collections are refreshed, the **Procedure** will no longer be in the **Views** collection and will appear in the **Procedures** collection.

See Also

[Views Append Method Example \(VB\)](#)

[Append Method \(Columns\)](#) | [Append Method \(Groups\)](#) | [Append Method \(Indexes\)](#) | [Append Method \(Keys\)](#) | [Append Method \(Procedures\)](#) | [Append Method \(Tables\)](#) | [Append Method \(Users\)](#)

Applies To: [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ChangePassword Method

Changes the password for a user account.

Syntax

```
User.ChangePassword OldPassword, NewPassword
```

Parameters

OldPassword

A **String** value that specifies the user's existing password. If the user doesn't currently have a password, use an empty string ("") for *OldPassword*.

NewPassword

A **String** value that specifies the new password.

Remarks

For security reasons, the old password must be specified in addition to the new password.

An error will occur if the provider does not support the administration of trustee properties.

See Also

[Groups and Users Append, ChangePassword Methods Example \(VB\)](#)

Applies To: [User Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Create Method

Creates a new catalog.

Syntax

```
Catalog.Create ConnectionString
```

Parameters

ConnectionString

A **String** value used to connect to the data source.

Remarks

The **Create** method creates and opens a new ADO [Connection](#) to the data source specified in *ConnectionString*. If successful, the new **Connection** object is assigned to the [ActiveConnection](#) property.

An error will occur if the [provider](#) does not support creating new catalogs.

See Also

[Create Method Example \(VB\)](#)

[ActiveConnection Property](#)

Applies To: [Catalog Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

Delete Method (Collections)

Removes an object from a collection.

Syntax

Collection.Delete *Name*

Parameters

Name

A **VARIANT** that specifies the name or ordinal position (index) of the object to delete.

Remarks

An error will occur if the *Name* does not exist in the collection.

For [Tables](#) and [Users](#) collections, an error will occur if the provider does not support deleting tables or users, respectively. For [Procedures](#) and [Views](#) collections, **Delete** will fail if the provider does not support persisting commands.

See Also

[Procedures Delete Method Example \(VB\)](#) | [Views Delete Method Example \(VB\)](#)

Applies To: [Columns Collection](#) | [Groups Collection](#) | [Indexes Collection](#) | [Keys Collection](#) | [Procedures Collection](#) | [Tables Collection](#) | [Users Collection](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

GetObjectOwner Method

Returns the owner of an object in a [Catalog](#).

Syntax

```
Owner = Catalog.GetObjectOwner(ObjectName, ObjectType [, ObjectTypeId])
```

Return Value

Returns a **String** value that specifies the [Name](#) of the [User](#) or [Group](#) that owns the object.

Parameters

ObjectName

A **String** value that specifies the name of the object for which to return the owner.

ObjectType

A **Long** value which can be one of the [ObjectTypeEnum](#) constants, that specifies the type of the object for which to get the owner.

ObjectTypeId

Optional. A **Variant** value that specifies the GUID for a provider object type not defined by the OLE DB specification. This parameter is required if *ObjectType* is set to **adPermObjProviderSpecific**; otherwise, it is not used.

Remarks

An error will occur if the provider does not support returning object owners.

See Also

[GetObjectOwner and SetObjectOwner Methods Example \(VB\)](#)

[SetObjectOwner Method](#)

Applies To: [Catalog Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

GetPermissions Method

Returns the permissions for a group or user on an object or object container.

Syntax

```
ReturnValue = GroupOrUser.GetPermissions(Name, ObjectType  
[, ObjectTypeId])
```

Return Value

Returns a **Long** value that specifies a bitmask containing the permissions that the group or user has on the object. This value can be one or more of the [RightsEnum](#) constants.

Parameters

Name

A **Variant** value that specifies the name of the object for which to set permissions. Set *Name* to a null value if you want to get the permissions for the object container.

ObjectType

A **Long** value which can be one of the [ObjectTypeEnum](#) constants, that specifies the type of the object for which to get permissions.

ObjectId

Optional. A **Variant** value that specifies the GUID for a provider object type not defined by the OLE DB specification. This parameter is required if *ObjectType* is set to **adPermObjProviderSpecific**; otherwise, it is not used.

See Also

[GetPermissions and SetPermissions Methods Example \(VB\)](#)

[Name Property](#) | [SetPermissions Method](#)

Applies To: [Group Object](#) | [User Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

SetObjectOwner Method

Specifies the owner of an object in a [Catalog](#).

Syntax

```
Catalog.SetObjectOwner ObjectName, ObjectType , OwnerName [, ObjectID]
```

Parameters

ObjectName

A **String** value that specifies the name of the object for which to specify the owner.

ObjectType

A **Long** value which can be one of the [ObjectTypeEnum](#) constants that specifies the owner type.

OwnerName

A **String** value that specifies the [Name](#) of the [User](#) or [Group](#) to own the object.

ObjectTypeId

Optional. A **Variant** value that specifies the GUID for a provider object type not defined by the OLE DB specification. This parameter is required if *ObjectType* is set to **adPermObjProviderSpecific**; otherwise, it is not used.

Remarks

An error will occur if the provider does not support specifying object owners.

See Also

[GetObjectOwner and SetObjectOwner Methods Example \(VB\)](#)

[GetObjectOwner Method](#)

Applies To: [Catalog Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 API Reference

SetPermissions Method

Specifies the permissions for a group or user on an object.

Syntax

```
GroupOrUser.SetPermissions Name, ObjectType, Action, Rights [, Inher  
[, ObjectId]
```

Parameters

Name

A **String** value that specifies the name of the object for which to set permissions.

ObjectType

A **Long** value which can be one of the [ObjectTypeEnum](#) constants, that specifies the type of the object for which to get permissions.

Action

A **Long** value which can be one of the [ActionEnum](#) constants that specifies the type of action to perform when setting permissions.

Rights

A **Long** value which can be a bitmask of one or more of the [RightsEnum](#) constants, that indicates the rights to set.

Inherit

Optional. A **Long** value which can be one of the [InheritTypeEnum](#) constants, that specifies how objects will inherit these permissions. The default value is **adInheritNone**.

ObjectId

Optional. A **Variant** value that specifies the GUID for a [provider](#) object type not defined by the OLE DB specification. This parameter is required if *ObjectType* is set to **adPermObjProviderSpecific**; otherwise, it is not used.

Remarks

An error will occur if the provider does not support setting access rights for groups or users.

Note When calling **SetPermissions**, setting Actions to **adAccessRevoke** overrides any settings of the *Rights* parameter. Do not set *Actions* to **adAccessRevoke** if you want the rights specified in the *Rights* parameter to take effect.

See Also

[GetPermissions and SetPermissions Methods Example \(VB\)](#)

[GetPermissions Method](#) | [Name Property](#)

Applies To: [Group Object](#) | [User Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ADOX Enumerated Constants

To assist debugging, the ADOX enumerated constants list a value for each constant. However, this value is purely advisory, and may change from one release of ADOX to another. Your code should only depend on the name, not the actual value, of enumerated constants.

The following enumerated constants are defined.

Enumeration	Description
ActionEnum	Specifies the type of action to be performed when SetPermissions is called.
AllowNullsEnum	Specifies whether records with null values are indexed.
ColumnAttributesEnum	Specifies characteristics of a Column .
DataTypeEnum	Specifies the data type of a Field , Parameter , or Property .
InheritTypeEnum	Specifies how objects will inherit permissions set with SetPermissions .
KeyTypeEnum	Specifies the type of Key : primary, foreign, or unique.
ObjectTypeEnum	Specifies the type of database object for which to set permissions or ownership.
RightsEnum	Specifies the rights or permissions for a group or user on an object.
RuleEnum	Specifies the rule to follow when a Key is deleted.
SortOrderEnum	Specifies the sort sequence for an indexed column.

See Also

[ADOX API Reference](#) | [Microsoft ADOX Programmer's Reference](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ActionEnum

Specifies the type of action to be performed when [SetPermissions](#) is called.

Constant	Value	Description
adAccessDeny	3	The group or user will be denied the specified permissions.
adAccessGrant	1	The group or user will have at least the requested permissions.
adAccessRevoke	4	Any explicit access rights the group or user has will be revoked.
adAccessSet	2	The group or user will have exactly the requested permissions.

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

AllowNullsEnum

Specifies whether records with null values are indexed.

Constant	Value	Description
adIndexNullsAllow	0	The index does allow entries in which the key columns are null. If a null value is entered in a key column, the entry is inserted into the index.
adIndexNullsDisallow	1	Default. The index does not allow entries in which the key columns are null. If a null value is entered in a key column, an error will occur.
adIndexNullsIgnore	2	The index does not insert entries containing null keys. If a null value is entered in a key column, the entry is ignored and no error occurs.
adIndexNullsIgnoreAny	4	The index does not insert entries where some key column has a null value. For an index having a multi-column key, if a null value is entered in some column, the entry is ignored and no error occurs.

See Also

Applies To: [IndexNulls Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ColumnAttributesEnum

Specifies characteristics of a [Column](#).

Constant	Value	Description
adColFixed	1	The column is a fixed length.
adColNullable	2	The column may contain null values.

See Also

Applies To: [Attributes Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

InheritTypeEnum

Specifies how objects will inherit permissions set with [SetPermissions](#).

Constant	Value	Description
adInheritBoth	3	Both objects and other containers contained by the primary object inherit the entry.
adInheritContainers	2	Other containers that are contained by the primary object inherit the entry.
adInheritNone	0	Default. No inheritance occurs.
adInheritNoPropagate	4	The adInheritObjects and adInheritContainers flags are not propagated to an inherited entry.
adInheritObjects	1	Non-container objects in the container inherit the permissions.

See Also

Applies To: [SetPermissions Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

KeyTypeEnum

Specifies the type of [Key](#): primary, foreign, or unique.

Constant	Value	Description
adKeyPrimary	1	Default. The key is a primary key.
adKeyForeign	2	The key is a foreign key.
adKeyUnique	3	The key is unique.

See Also

Applies To: [Type Property \(Key\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

ObjectTypeEnum

Specifies the type of database object for which to set permissions or ownership.

Constant	Value	Description
adPermObjColumn	2	The object is a column.
adPermObjDatabase	3	The object is a database.
adPermObjProcedure	4	The object is a procedure.
adPermObjProviderSpecific -1		The object is a type defined by the provider . An error will occur if the <i>ObjectType</i> parameter is adPermObjProviderSpecific and an <i>ObjectTypeId</i> is not supplied.
adPermObjTable	1	The object is a table.
adPermObjView	5	The object is a view.

See Also

Applies To: [GetObjectOwner Method](#) | [GetPermissions Method](#) | [SetObjectOwner Method](#) | [SetPermissions Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

RightsEnum

Specifies the rights or permissions for a group or user on an object.

Constant	Value	Description
adRightCreate	16384 (&H4000)	The user or group has permission to create new objects of this type.
adRightDelete	65536 (&H10000)	The user or group has permission to delete data from an object. For objects such as Tables , the user has permission to delete data values from records.
adRightDrop	256 (&H100)	The user or group has permission to remove objects from the catalog. For example, Tables can be deleted by a DROP TABLE SQL command.
adRightExclusive	512 (&H200)	The user or group has permission to access the object exclusively.
adRightExecute	536870912 (&H20000000)	The user or group has permission to execute the object.
adRightFull	268435456 (&H10000000)	The user or group has all permissions on the object.
adRightInsert	32768 (&H8000)	The user or group has permission to insert the object. For objects such as Tables , the user has permission to insert data into the table.
adRightMaximumAllowed	33554432 (&H2000000)	The user or group has the maximum number of permissions allowed by the provider. Specific permissions are provider-dependent.
adRightNone	0	The user or group has no permissions for the object.

adRightRead	-2147483648 (&H80000000)	The user or group has permission to read the object. For objects such as Tables , the user has permission to read the data in the table.
adRightReadDesign	1024 (&H400)	The user or group has permission to read the design for the object.
adRightReadPermissions	131072 (&H20000)	The user or group can view, but not change, the specific permissions for an object in the catalog.
adRightReference	8192 (&H2000)	The user or group has permission to reference the object.
adRightUpdate	1073741824 (&H40000000)	The user or group has permission to update the object. For objects such as Tables , the user has permission to update the data in the table.
adRightWithGrant	4096 (&H1000)	The user or group has permission to grant permissions on the object.
adRightWriteDesign	2048 (&H800)	The user or group has permission to modify the design for the object.
adRightWriteOwner	524288 (&H80000)	The user or group has permission to modify the owner of the object.
adRightWritePermissions	262144 (&H40000)	The user or group can modify the specific permissions for an object in the catalog.

See Also

Applies To: [GetPermissions Method](#) | [SetPermissions Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

RuleEnum

Specifies the rule to follow when a [Key](#) is deleted.

Constant	Value	Description
adRICascade	1	Cascade changes.
adRINone	0	Default. No action is taken.
adRISetDefault	3	Foreign key value is set to the default.
adRISetNull	2	Foreign key value is set to null.

See Also

Applies To: [DeleteRule Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 API Reference

SortOrderEnum

Specifies the sort sequence for an indexed column.

Constant	Value	Description
adSortAscending	1	Default. The sort sequence for the column is ascending.
adSortDescending	2	The sort sequence for the column is descending.

See Also

Applies To: [SortOrder Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

ADOX Code Examples

Use the following code examples to learn how to use the ADOX objects, methods, properties, and events.

Note Paste the entire code example into your code editor. The example may not run correctly if partial examples are used or if paragraph formatting is lost.

- [ADOX Code Examples in Microsoft Visual Basic](#)
- [ADOX Code Examples in Microsoft Visual C++](#)

See Also

[Catalog ActiveConnection Property Example \(VB\)](#) | [Columns and Tables Append Methods, Name Property Example \(VB\)](#) | [Command and CommandText Properties Example \(VB\)](#) | [Connection Close Method, Table Type Property Example \(VB\)](#) | [GetPermissions and SetPermissions Methods Example \(VB\)](#) | [Indexes Append Method Example \(VB\)](#) | [Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example \(VB\)](#) | [Microsoft ADOX Programmer's Reference](#) | [Parameters Collection, Command Property Example \(VB\)](#) | [ParentCatalog Property Example \(VB\)](#) | [Procedures Append Method Example \(VB\)](#) | [Procedures Delete Method Example \(VB\)](#) | [Procedures Refresh Method Example \(VB\)](#) | [Views and Fields Collections Example \(VB\)](#) | [Views Append Method Example \(VB\)](#) | [Views Collection, CommandText Property Example \(VB\)](#) | [Views Delete Method Example \(VB\)](#) | [Views Refresh Method Example \(VB\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

ADOX Code Examples in Microsoft Visual Basic

These topics provide sample code to help you understand how to use ADOX. All code examples are written using Microsoft Visual Basic.

Note Paste the entire code example, from Sub to End Sub, into your code editor. The example may not run correctly if you use partial examples or if paragraph formatting is lost.

Methods

- [Columns and Tables Append Methods, Name Property Example \(VB\)](#)
- [Connection Close Method, Table Type Property Example \(VB\)](#)
- [Create Method Example \(VB\)](#)
- [GetObjectOwner and SetObjectOwner Methods Example \(VB\)](#)
- [GetPermissions and SetPermissions Methods Example \(VB\)](#)
- [Groups and Users Append, ChangePassword Methods Example \(VB\)](#)
- [Indexes Append Method Example \(VB\)](#)
- [Keys Append Method, Key Type, RelatedColumn, RelatedTable, and UpdateRule Properties Example \(VB\)](#)
- [Procedures Append Method Example \(VB\)](#)
- [Procedures Delete Method Example \(VB\)](#)
- [Procedures Refresh Method Example \(VB\)](#)
- [Views Append Method Example \(VB\)](#)
- [Views Delete Method Example \(VB\)](#)
- [Views Refresh Method Example \(VB\)](#)

Properties

- [Attributes Property Example \(VB\)](#)
- [Catalog ActiveConnection Property Example \(VB\)](#)
- [Clustered Property Example \(VB\)](#)
- [Command and CommandText Properties Example \(VB\)](#)
- [Command Property, Parameters Collection Example \(VB\)](#)
- [CommandText Property, Views Collection Example \(VB\)](#)

- [DateCreated and DateModified Properties Example \(VB\)](#)
- [DefinedSize Property Example \(VB\)](#)
- [DeleteRule Property Example \(VB\)](#)
- [IndexNulls Property Example \(VB\)](#)
- [Key Type, RelatedColumn, RelatedTable, and UpdateRule Properties, Keys Append Method Example \(VB\)](#)
- [Name Property, Columns and Tables Append Methods Example \(VB\)](#)
- [NumericScale and Precision Properties Example \(VB\)](#)
- [ParentCatalog Property Example \(VB\)](#)
- [PrimaryKey and Unique Properties Example \(VB\)](#)
- [SortOrder Property Example \(VB\)](#)
- [Table Type Property, Connection Close Method, Example \(VB\)](#)

Collections

- [Parameters Collection, Command Property Example \(VB\)](#)
- [Views and Fields Collections Example \(VB\)](#)
- [Views Collection, CommandText Property Example \(VB\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Attributes Property Example (VB)

This example demonstrates the [Attributes](#) property of a [Column](#). Setting it to **adColNullable** allows the user to set the value of a [Recordset Field](#) to an empty string. In this situation, the user can distinguish between a record where data is not known and a record where the data does not apply.

```
' BeginAttributesVB
Sub Main()
    On Error GoTo AttributesXError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim colTemp As New ADOX.Column
    Dim rstEmployees As New Recordset
    Dim strMessage As String
    Dim strInput As String
    Dim tblEmp As ADOX.Table

    ' Connect the catalog.
    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';data source=" & _
        "'c:\Program Files\Microsoft Office\Office\Samples\Northwind
    Set cat.ActiveConnection = cnn

    Set tblEmp = cat.Tables("Employees")

    ' Create a new Field object and append it to the Fields
    ' collection of the Employees table.
    colTemp.Name = "FaxPhone"
    colTemp.Type = adVarChar
    colTemp.DefinedSize = 24
    colTemp.Attributes = adColNullable
    cat.Tables("Employees").Columns.Append colTemp.Name, adVarChar, 24

    ' Open the Employees table for updating as a Recordset
    rstEmployees.Open "Employees", cnn, adOpenKeyset, adLockOptimist

    With rstEmployees
        ' Get user input.
        strMessage = "Enter fax number for " & _
            !FirstName & " " & !LastName & "." & vbCrLf & _
            "[? - unknown, X - has no fax]"
        strInput = UCase(InputBox(strMessage))
        If strInput <> "" Then
            Select Case strInput
```

```

        Case "?"
            !FaxPhone = Null
        Case "X"
            !FaxPhone = ""
        Case Else
            !FaxPhone = strInput
    End Select
    .Update

    ' Print report.
    Debug.Print "Name - Fax number"
    Debug.Print !FirstName & " " & !LastName & " - ";

    If IsNull(!FaxPhone) Then
        Debug.Print "[Unknown]"
    Else
        If !FaxPhone = "" Then
            Debug.Print "[Has no fax]"
        Else
            Debug.Print !FaxPhone
        End If
    End If

    End If

    .Close
End With

'Clean up
tblEmp.Columns.Delete colTemp.Name
cnn.Close
Set rstEmployees = Nothing
Set cat = Nothing
Set colTemp = Nothing
Set cnn = Nothing
Exit Sub

```

AttributesXError:

```

If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Not tblEmp Is Nothing Then tblEmp.Columns.Delete colTemp.Name

Set cat = Nothing
Set colTemp = Nothing

If Not cnn Is Nothing Then

```

```
        If cnn.State = adStateOpen Then cnn.Close
    End If
    Set cnn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If

End Sub
' EndAttributesVB
```

See Also

[Attributes Property](#) | [Catalog Object](#) | [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Catalog ActiveConnection Property Example (VB)

Setting the [ActiveConnection](#) property to a valid, open connection "opens" the catalog. From an open catalog, you can access the schema objects contained within that catalog.

```
' BeginOpenConnectionVB
Sub OpenConnection()
    On Error GoTo OpenConnectionError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog

    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source= 'c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"
    Set cat.ActiveConnection = cnn
    Debug.Print cat.Tables(0).Type

    'Clean up
    cnn.Close
    Set cat = Nothing
    Set cnn = Nothing
    Exit Sub

OpenConnectionError:

    Set cat = Nothing

    If Not cnn Is Nothing Then
        If cnn.State = adStateOpen Then cnn.Close
    End If
    Set cnn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndOpenConnectionVB
```

Setting the **ActiveConnection** property to a valid connection string also "opens" the catalog.

```

' BeginOpenConnection2VB
Sub Main()
    On Error GoTo OpenConnectionWithStringError
    Dim cat As New ADOX.Catalog

    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"
    Debug.Print cat.Tables(0).Type

    'Clean up
    Set cat.ActiveConnection = Nothing
    Exit Sub

OpenConnectionWithStringError:
    Set cat = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndOpenConnection2VB

```

See Also

[ActiveConnection Property](#) |
 [Catalog Object](#) |
 [Table Object](#) |
 [Tables Collection](#) |
 [Type Property \(Table\)](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 

Clustered Property Example (VB)

This example demonstrates the [Clustered](#) property of an [Index](#). Note that Microsoft Jet databases do not support clustered indexes, so this example will return **False** for the **Clustered** property of all indexes in the *Northwind* database.

```
' BeginClusteredVB
Sub Main()
    On Error GoTo ClusteredXError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim tblLoop As ADOX.Table
    Dim idxLoop As ADOX.Index
    Dim strCnn As String

    strCnn = "Provider='SQLOLEDB';Data Source='MySqlServer';Initial
        "Integrated Security='SSPI';"
    ' Connect the catalog.
    cnn.Open strCnn
    cat.ActiveConnection = cnn

    ' Enumerate Tables
    For Each tblLoop In cat.Tables
        'Enumerate Indexes
        For Each idxLoop In tblLoop.Indexes
            Debug.Print tblLoop.Name & " " & _
                idxLoop.Name & " " & idxLoop.Clustered
        Next idxLoop
    Next tblLoop

    'Clean up
    cnn.Close
    Set cat = Nothing
    Set cnn = Nothing
    Exit Sub
```

ClusteredXError:

```
Set cat = Nothing

If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing
```

```
    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndClusteredVB
```

See Also

[Catalog Object](#) | [Clustered Property](#) | [Index Object](#) | [Table Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Columns and Tables Append Methods, Name Property Example (VB)

The following code demonstrates how to create a new table.

```
' BeginCreateTableVB
Sub Main()
    On Error GoTo CreateTableError

    Dim tbl As New Table
    Dim cat As New ADOX.Catalog

    ' Open the Catalog.
    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    tbl.Name = "MyTable"
    tbl.Columns.Append "Column1", adInteger
    tbl.Columns.Append "Column2", adInteger
    tbl.Columns.Append "Column3", adVarChar, 50
    cat.Tables.Append tbl
    Debug.Print "Table 'MyTable' is added."

    'Delete the table as this is a demonstration.
    cat.Tables.Delete tbl.Name
    Debug.Print "Table 'MyTable' is deleted."

    'Clean up
    Set cat.ActiveConnection = Nothing
    Set cat = Nothing
    Set tbl = Nothing
    Exit Sub

CreateTableError:

    Set cat = Nothing
    Set tbl = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
```

```
End If  
End Sub  
' EndCreateTableVB
```

See Also

[Append Method \(Columns\)](#) | [Append Method \(Tables\)](#) | [Column Object](#) | [Columns Collection](#) | [Name Property](#) | [Table Object](#) | [Tables Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Command and CommandText Properties Example (VB)

The following code demonstrates how to use the [Command](#) property to update the text of a procedure.

```
' BeginProcedureTextVB
Sub Main()
    On Error GoTo ProcedureTextError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim cmd As New ADODB.Command

    ' Open the Connection
    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Open the catalog
    Set cat.ActiveConnection = cnn

    ' Get the Command
    Set cmd = cat.Procedures("CustomerById").Command

    ' Update the CommandText
    cmd.CommandText = "Select CustomerId, CompanyName, ContactName " & _
        "From Customers " & _
        "Where CustomerId = [CustId]"

    ' Update the Procedure
    Set cat.Procedures("CustomerById").Command = cmd

    'Clean up
    cnn.Close
    Set cat = Nothing
    Set cmd = Nothing
    Set cnn = Nothing
    Exit Sub

ProcedureTextError:
    Set cat = Nothing
    Set cmd = Nothing
```

```
If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
' EndProcedureTextVB
```

See Also

[ActiveConnection Property](#) | [Catalog Object](#) | [Command Property](#) | [Procedure Object](#) | [Procedures Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Connection Close Method, Table Type Property Example (VB)

Setting the [ActiveConnection](#) property to **Nothing** should "close" the catalog. Associated collections will be empty. Any objects that were created from schema objects in the catalog will be orphaned. Any properties on those objects that have been cached will still be available, but attempting to read properties that require a call to the provider will fail.

```
' BeginCloseConnectionVB
Sub Main()
    On Error GoTo CloseConnectionByNothingError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim tbl As ADOX.Table

    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source= 'c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"
    Set cat.ActiveConnection = cnn
    Set tbl = cat.Tables(0)
    Debug.Print tbl.Type      ' Cache tbl.Type info
    Set cat.ActiveConnection = Nothing
    Debug.Print tbl.Type      ' tbl is orphaned
    ' Previous line will succeed if this was cached
    Debug.Print tbl.Columns(0).DefinedSize
    ' Previous line will fail if this info has not been cached

    'Clean up
    cnn.Close
    Set cat = Nothing
    Set cnn = Nothing
    Exit Sub

CloseConnectionByNothingError:
    Set cat = Nothing

    If Not cnn Is Nothing Then
        If cnn.State = adStateOpen Then cnn.Close
    End If
    Set cnn = Nothing
```

```

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndCloseConnectionVB

```

Closing a [Connection](#) object that was used to "open" the catalog should have the same effect as setting the **ActiveConnection** property to **Nothing**.

```

Sub CloseConnection()
    On Error GoTo CloseConnectionError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim tbl As ADOX.Table

    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source= 'c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb'";"
    Set cat.ActiveConnection = cnn
    Set tbl = cat.Tables(0)
    Debug.Print tbl.Type      ' Cache tbl.Type info
    cnn.Close
    Debug.Print tbl.Type      ' tbl is orphaned
    ' Previous line will succeed if this was cached
    Debug.Print tbl.Columns(0).DefinedSize
    ' Previous line will fail if this info has not been cached

    'Clean up
    Set cat = Nothing
    Set cnn = Nothing
    Exit Sub

```

CloseConnectionError:

```

    Set cat = Nothing

    If Not cnn Is Nothing Then
        If cnn.State = adStateOpen Then cnn.Close
    End If
    Set cnn = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndCloseConnection2VB

```

See Also

[ActiveConnection Property](#) | [Catalog Object](#) | [Column Object](#) | [Columns Collection](#) | [Table Object](#) | [Tables Collection](#) | [Type Property \(Table\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Create Method Example (VB)

The following code shows how to create a new Microsoft Jet database with the [Create](#) method.

```
' BeginCreateDatabaseVB
Sub CreateDatabase()
    On Error GoTo CreateDatabaseError

    Dim cat As New ADOX.Catalog
    cat.Create "Provider='Microsoft.Jet.OLEDB.4.0';Data Source='c:\n

    'Clean up
    Set cat = Nothing
    Exit Sub

CreateDatabaseError:
    Set cat = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndCreateDatabaseVB
```

See Also

[Catalog Object](#) | [Create Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

DateCreated and DateModified Properties Example (VB)

This example demonstrates the [DateCreated](#) and [DateModified](#) properties by adding a new [Column](#) to an existing [Table](#) and by creating a new **Table**. The DateOutput procedure is required for this example to run.

```
' BeginDateCreatedVB
Sub Main()
    On Error GoTo DateCreatedXError

    Dim cat As New ADOX.Catalog
    Dim tblEmployees As ADOX.Table
    Dim tblNewTable As ADOX.Table

    ' Connect the catalog.
    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';"

    With cat
        Set tblEmployees = .Tables("Employees")

        ' Print current information about the Employees table.
        DateOutput "Current properties", tblEmployees

        ' Create and append column to the Employees table.
        tblEmployees.Columns.Append "NewColumn", adInteger
        .Tables.Refresh

        ' Print new information about the Employees table.
        DateOutput "After creating a new column", tblEmployees

        ' Delete new column because this is a demonstration.
        tblEmployees.Columns.Delete "NewColumn"

        ' Create and append new Table object to the Northwind databa
        Set tblNewTable = New ADOX.Table
        tblNewTable.Name = "NewTable"
        tblNewTable.Columns.Append "NewColumn", adInteger
        .Tables.Append tblNewTable
        .Tables.Refresh
```

```

        ' Print information about the new Table object.
        DateOutput "After creating a new table", .Tables("NewTable")

        ' Delete new Table object because this is a demonstration.
        .Tables.Delete tblNewTable.Name
    End With

    'Clean up
    Set cat.ActiveConnection = Nothing
    Set cat = Nothing
    Exit Sub

DateCreatedXError:
    Set cat = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If

End Sub

Sub DateOutput(strTemp As String, tblTemp As ADOX.Table)
    ' Print DateCreated and DateModified information about
    ' specified Table object.
    Debug.Print strTemp
    Debug.Print "    Table: " & tblTemp.Name
    Debug.Print "        DateCreated = " & tblTemp.DateCreated
    Debug.Print "        DateModified = " & tblTemp.DateModified
    Debug.Print
End Sub
' EndDateCreatedVB

```

See Also

[DateCreated Property](#) | [DateModified Property](#) | [Procedure Object](#) | [Procedures Collection](#) | [View Object](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

DefinedSize Property Example (VB)

This example demonstrates the [DefinedSize](#) property of a [Column](#). The code will redefine the size of the FirstName column of the **Employees** table of the *Northwind* database. Then, the change in the values of the FirstName [Field](#) of a [Recordset](#) based on the **Employees** table is displayed. Note that by default, the FirstName field becomes padded with spaces after you redefine the **DefinedSize** property.

```
' BeginDefinedSizeVB
Public Sub Main()
    On Error GoTo DefinedSizeXError

    Dim rstEmployees As ADODB.Recordset
    Dim catNorthwind As New ADOX.Catalog
    Dim colFirstName As ADOX.Column
    Dim colNewFirstName As New ADOX.Column
    Dim aryFirstName() As String
    Dim i As Integer
    Dim strCnn As String

    strCnn = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
            "Data Source='c:\Program Files\" & _
            "Microsoft Office\Office\Samples\Northwind.mdb';"

    ' Open a Recordset for the Employees table.
    Set rstEmployees = New ADODB.Recordset
    rstEmployees.Open "Employees", strCnn, adOpenKeyset, , adCmdTable
    ReDim aryFirstName(rstEmployees.RecordCount)

    ' Open a Catalog for the Northwind database,
    ' using same connection as rstEmployees
    Set catNorthwind.ActiveConnection = rstEmployees.ActiveConnection

    ' Loop through the recordset displaying the contents
    ' of the FirstName field, the field's defined size,
    ' and its actual size.
    ' Also store FirstName values in aryFirstName array.
    rstEmployees.MoveFirst
    Debug.Print " "
    Debug.Print "Original Defined Size and Actual Size"
    i = 0
    Do Until rstEmployees.EOF
        Debug.Print "Employee name: " & rstEmployees!FirstName & _
```

```

        " " & rstEmployees!LastName
    Debug.Print "    FirstName Defined size: " _
        & rstEmployees!FirstName.DefinedSize
    Debug.Print "    FirstName Actual size: " & _
        rstEmployees!FirstName.ActualSize
    If Not rstEmployees!FirstName = Null Then
        aryFirstName(i) = rstEmployees!FirstName
    End If
    rstEmployees.MoveNext
    i = i + 1
Loop
rstEmployees.Close

' Redefine the DefinedSize of FirstName in the catalog
Set colFirstName = catNorthwind.Tables("Employees").Columns("Fir
colNewFirstName.Name = colFirstName.Name
colNewFirstName.Type = colFirstName.Type
colNewFirstName.DefinedSize = colFirstName.DefinedSize + 1

' Append new FirstName column to catalog
catNorthwind.Tables("Employees").Columns.Delete colFirstName.Nam
catNorthwind.Tables("Employees").Columns.Append colNewFirstName

' Open Employee table in Recordset for updating
rstEmployees.Open "Employees", catNorthwind.ActiveConnection, _
    adOpenKeyset, adLockOptimistic, adCmdTable

' Loop through the recordset displaying the contents
' of the FirstName field, the field's defined size,
' and its actual size.
' Also restore FirstName values from aryFirstName.
rstEmployees.MoveFirst
Debug.Print " "
Debug.Print "New Defined Size and Actual Size"
i = 0
Do Until rstEmployees.EOF
    rstEmployees!FirstName = aryFirstName(i)
    Debug.Print "Employee name: " & rstEmployees!FirstName & _
        " " & rstEmployees!LastName
    Debug.Print "    FirstName Defined size: " _
        & rstEmployees!FirstName.DefinedSize
    Debug.Print "    FirstName Actual size: " & _
        rstEmployees!FirstName.ActualSize
    rstEmployees.MoveNext
    i = i + 1
Loop
rstEmployees.Close

' Restore original FirstName column to catalog
catNorthwind.Tables("Employees").Columns.Delete colNewFirstName.

```

```

catNorthwind.Tables("Employees").Columns.Append colFirstName

' Restore original FirstName values to Employees table
rstEmployees.Open "Employees", catNorthwind.ActiveConnection, _
    adOpenKeyset, adLockOptimistic, adCmdTable

rstEmployees.MoveFirst
i = 0
Do Until rstEmployees.EOF
    rstEmployees!FirstName = aryFirstName(i)
    rstEmployees.MoveNext
    i = i + 1
Loop
rstEmployees.Close

'Clean up
Set catNorthwind = Nothing
Set colNewFirstName = Nothing
Set colFirstName = Nothing
Set rstEmployees = Nothing
Exit Sub

DefinedSizeXError:
Set catNorthwind = Nothing
Set colNewFirstName = Nothing
Set colFirstName = Nothing

If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

End Sub
' EndDefinedSizeVB

```

See Also

[Column Object](#) | [DefinedSize Property](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 

DeleteRule Property Example (VB)

This example demonstrates the [DeleteRule](#) property of a [Key](#) object. The code appends a new [Table](#) and then defines a new primary key, setting **DeleteRule** to **adRiCascade**.

```
' BeginDeleteRuleVB
Sub Main()
    On Error GoTo DeleteRuleXError

    Dim kyPrimary As New ADOX.Key
    Dim cat As New ADOX.Catalog
    Dim tblNew As New ADOX.Table

    ' Connect the catalog
    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "data source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb'"

    ' Name new table
    tblNew.Name = "NewTable"

    ' Append a numeric and a text field to new table.
    tblNew.Columns.Append "NumField", adInteger, 20
    tblNew.Columns.Append "TextField", adVarChar, 20

    ' Append the new table
    cat.Tables.Append tblNew

    ' Define the Primary key
    kyPrimary.Name = "NumField"
    kyPrimary.Type = adKeyPrimary
    kyPrimary.RelatedTable = "Customers"
    kyPrimary.Columns.Append "NumField"
    kyPrimary.Columns("NumField").RelatedColumn = "CustomerId"
    kyPrimary.DeleteRule = adRiCascade

    ' Append the primary key
    cat.Tables("NewTable").Keys.Append kyPrimary
    Debug.Print "The primary key is appended."

    'Delete the table as this is a demonstration.
    cat.Tables.Delete tblNew.Name
    Debug.Print "The primary key is deleted."
```

```
'Clean up
Set cat.ActiveConnection = Nothing
Set cat = Nothing
Set kyPrimary = Nothing
Set tblNew = Nothing
Exit Sub
```

DeleteRuleXError:

```
Set cat = Nothing
Set kyPrimary = Nothing
Set tblNew = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
```

```
End Sub
' EndDeleteRuleVB
```

See Also

[DeleteRule Property](#) | [Key Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

GetObjectOwner and SetObjectOwner Methods Example (VB)

This example demonstrates the [GetObjectOwner](#) and [SetObjectOwner](#) methods. This code assumes the existence of the group Accounting (see the [Groups and Users Append, ChangePassword Methods Example \(VB\)](#) to see how to add this group to the system). The owner of the Categories table is set to Accounting.

```
' BeginOwnersVB
Sub Main()
    On Error GoTo OwnersXError

    Dim tblLoop As New ADOX.Table
    Dim cat As New ADOX.Catalog
    Dim strOwner As String

    ' Open the Catalog.
    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';" & _
        "jet oledb:system database=" & _
        "'c:\Program Files\Microsoft Office\Office\system.mdw'"

    ' Print the original owner of Categories
    strOwner = cat.GetObjectOwner("Categories", adPermObjTable)
    Debug.Print "Owner of Categories: " & strOwner

    ' Set the owner of Categories to Accounting
    cat.SetObjectOwner "Categories", adPermObjTable, "Accounting"

    ' List the owners of all tables and columns in the catalog.
    For Each tblLoop In cat.Tables
        Debug.Print "Table: " & tblLoop.Name
        Debug.Print "    Owner: " & _
            cat.GetObjectOwner(tblLoop.Name, adPermObjTable)
    Next tblLoop

    ' Restore the original owner of Categories
    cat.SetObjectOwner "Categories", adPermObjTable, strOwner
```

```
'Clean up
Set cat.ActiveConnection = Nothing
Set cat = Nothing
Exit Sub
```

OwnersXError:

```
Set cat = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
```

```
End Sub
' EndOwnersVB
```

See Also

[Catalog Object](#) | [GetObjectOwner Method](#) | [SetObjectOwner Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

GetPermissions and SetPermissions Methods Example (VB)

This example demonstrates the [GetPermissions](#) and [SetPermissions](#) methods. The following code gives full access for the Orders table to the Admin user.

```
' BeginGrantPermissionsVB
Sub GrantPermissions()
    On Error GoTo GrantPermissionsError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim lngPerm As Long

    ' Opens a connection to the northwind database
    ' using the Microsoft Jet 4.0 provider
    cnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    cnn.Open "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';" & _
        "jet oledb:system database=" & _
        "'c:\Program Files\Microsoft Office\Office\system.mdw'"

    Set cat.ActiveConnection = cnn

    ' Retrieve original permissions
    lngPerm = cat.Users("admin").GetPermissions("Orders", adPermObjT
    Debug.Print "Original permissions: " & Str(lngPerm)

    ' Revoke all permissions
    cat.Users("admin").SetPermissions "Orders", adPermObjTable, _
        adAccessRevoke, adRightFull

    ' Display permissions
    Debug.Print "Revoked permissions: " & _
        Str(cat.Users("admin").GetPermissions("Orders", adPermObjTab

    ' Give the Admin user full rights on the orders object
    cat.Users("admin").SetPermissions "Orders", adPermObjTable, _
        adAccessSet, adRightFull

    ' Display permissions
    Debug.Print "Full permissions: " & _
        Str(cat.Users("admin").GetPermissions("Orders", adPermObjTab
```

```
' Restore original permissions
cat.Users("admin").SetPermissions "Orders", adPermObjTable, _
    adAccessSet, lngPerm

' Display permissions
Debug.Print "Final permissions: " & _
    Str(cat.Users("admin").GetPermissions("Orders", adPermObjTab

'Clean up
cnn.Close
Set cat = Nothing
Set cnn = Nothing
Exit Sub
```

GrantPermissionsError:

```
Set cat = Nothing

If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
```

End Sub

' **EndGrantPermissionsVBSee Also**

[Catalog Object](#) | [GetPermissions Method](#) | [SetPermissions Method](#) | [User Object](#) | [Users Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Groups and Users Append, ChangePassword Methods Example (VB)

This example demonstrates the [Append](#) method of [Groups](#), as well as the [Append](#) method of [Users](#) by adding a new [Group](#) and a new [User](#) to the system. The new **Group** is appended to the **Groups** collection of the new **User**. Consequently, the new **User** is added to the **Group**. Also, the [ChangePassword](#) method is used to specify the **User** password.

```
' BeginGroupVB
Sub Main()
    On Error GoTo GroupXError

    Dim cat As ADOX.Catalog
    Dim usrNew As ADOX.User
    Dim usrLoop As ADOX.User
    Dim grpLoop As ADOX.Group

    Set cat = New ADOX.Catalog

    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';" & _
        "jet oledb:system database=" & _
        "'c:\Program Files\Microsoft Office\Office\system.mdw'"

    With cat
        'Create and append new group with a string.
        .Groups.Append "Accounting"

        ' Create and append new user with an object.
        Set usrNew = New ADOX.User
        usrNew.Name = "Pat Smith"
        usrNew.ChangePassword "", "Password1"
        .Users.Append usrNew

        ' Make the user Pat Smith a member of the
        ' Accounting group by creating and adding the
        ' appropriate Group object to the user's Groups
        ' collection. The same is accomplished if a User
```

```

' object representing Pat Smith is created and
' appended to the Accounting group Users collection
usrNew.Groups.Append "Accounting"

' Enumerate all User objects in the
' catalog's Users collection.
For Each usrLoop In .Users
    Debug.Print " " & usrLoop.Name
    Debug.Print "     Belongs to these groups:"
    ' Enumerate all Group objects in each User
    ' object's Groups collection.
    If usrLoop.Groups.Count <> 0 Then
        For Each grpLoop In usrLoop.Groups
            Debug.Print "     " & grpLoop.Name
        Next grpLoop
    Else
        Debug.Print "     [None]"
    End If
Next usrLoop

' Enumerate all Group objects in the default
' workspace's Groups collection.
For Each grpLoop In .Groups
    Debug.Print " " & grpLoop.Name
    Debug.Print "     Has as its members:"
    ' Enumerate all User objects in each Group
    ' object's Users collection.
    If grpLoop.Users.Count <> 0 Then
        For Each usrLoop In grpLoop.Users
            Debug.Print "     " & usrLoop.Name
        Next usrLoop
    Else
        Debug.Print "     [None]"
    End If
Next grpLoop

' Delete new User and Group objects because this
' is only a demonstration.
' These two line are commented out because the sub "OwnersX"
' the group "Accounting".
'
' .Users.Delete "Pat Smith"
' .Groups.Delete "Accounting"

End With

'Clean up
Set cat.ActiveConnection = Nothing
Set cat = Nothing
Set usrNew = Nothing
Exit Sub

```

GroupXError:

```
    Set cat = Nothing
    Set usrNew = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndGroupVB
```

See Also

[Append Method \(Groups\)](#) | [Append Method \(Users\)](#) | [Catalog Object](#) | [ChangePassword Method](#) | [Group Object](#) | [Groups Collection](#) | [User Object](#) | [Users Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Indexes Append Method Example (VB)

The following code demonstrates how to create a new index. The index is on two columns in the table.

```
' BeginCreateIndexVB
Sub Main()
    On Error GoTo CreateIndexError

    Dim tbl As New Table
    Dim idx As New ADOX.Index
    Dim cat As New ADOX.Catalog

    'Open the catalog.
    ' Open the Catalog.
    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Define the table and append it to the catalog
    tbl.Name = "MyTable"
    tbl.Columns.Append "Column1", adInteger
    tbl.Columns.Append "Column2", adInteger
    tbl.Columns.Append "Column3", adVarChar, 50
    cat.Tables.Append tbl
    Debug.Print "Table 'MyTable' is added."

    ' Define a multi-column index
    idx.Name = "multicolidx"
    idx.Columns.Append "Column1"
    idx.Columns.Append "Column2"

    ' Append the index to the table
    tbl.Indexes.Append idx
    Debug.Print "The index is appended to table 'MyTable'."

    'Delete the table as this is a demonstration
    cat.Tables.Delete tbl.Name
    Debug.Print "Table 'MyTable' is deleted."

    'Clean up
    Set cat.ActiveConnection = Nothing
    Set cat = Nothing
```

```
Set tbl = Nothing
Set idx = Nothing
Exit Sub
```

```
CreateIndexError:
```

```
Set cat = Nothing
Set tbl = Nothing
Set idx = Nothing
```

```
If Err <> 0 Then
```

```
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
```

```
End If
```

```
End Sub
```

```
' EndCreateIndexVB
```

See Also

[Append Method \(Indexes\)](#) | [Index Object](#) | [Indexes Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

IndexNulls Property Example (VB)

This example demonstrates the [IndexNulls](#) property of an [Index](#). The code creates a new index and sets the value of **IndexNulls** based on user input (from a list box named List1). Then, the **Index** is appended to the **Employees Table** in the *Northwind Catalog*. The new **Index** is applied to a [Recordset](#) based on the **Employees** table, and the **Recordset** is opened. A new record is added to the **Employees** table, with a **Null** value in the indexed field. Whether this new record is displayed depends on the setting of the **IndexNulls** property.

```
' IndexNullsVB
Sub Main()
    On Error GoTo IndexNullsXError

    Dim cnn As New ADODB.Connection
    Dim catNorthwind As New ADOX.Catalog
    Dim idxNew As New ADOX.Index
    Dim rstEmployees As New ADODB.Recordset
    Dim varBookmark As Variant

    ' Connect the catalog.
    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb";"

    Set catNorthwind.ActiveConnection = cnn

    ' Append Country column to new index
    idxNew.Columns.Append "Country"
    idxNew.Name = "NewIndex"

    Dim Response
    Response = MsgBox("Allow 'Null' index? Otherwise ignore 'Null' i
        \"Allow 'Null' index? Otherwise ignore 'Null' index.\"
        , vbYesNo + vbCritical + vbDefaultButton2,,,,
    If Response = vbYes Then ' User chose Yes.
        idxNew.IndexNulls = adIndexNullsAllow
    Else ' User chose No.
        idxNew.IndexNulls = adIndexNullsIgnore
    End If

    'Append new index to Employees table
    catNorthwind.Tables("Employees").Indexes.Append idxNew
```

```

rstEmployees.Index = idxNew.Name
rstEmployees.Open "Employees", cnn, adOpenKeyset, _
    adLockOptimistic, adCmdTableDirect

With rstEmployees
    ' Add a new record to the Employees table.
    .AddNew
    !FirstName = "Gary"
    !LastName = "Haarsager"
    .Update

    ' Bookmark the newly added record
    varBookmark = .Bookmark

    ' Use the new index to set the order of the records.
    .MoveFirst

    Debug.Print "Index = " & .Index & _
        ", IndexNulls = " & idxNew.IndexNulls
    Debug.Print "  Country - Name"

    ' Enumerate the Recordset. The value of the
    ' IndexNulls property will determine if the newly
    ' added record appears in the output.
    Do While Not .EOF
        Debug.Print "      " & _
            IIf(IsNull(!Country), "[Null]", !Country) & _
            " - " & !FirstName & " " & !LastName
        .MoveNext
    Loop

    ' Delete new record because this is a demonstration.
    .Bookmark = varBookmark
    .Delete

    .Close
End With

'Clean up
Set rstEmployees = Nothing
catNorthwind.Tables("Employees").Indexes.Delete idxNew.Name
cnn.Close
Set cnn = Nothing
Set catNorthwind = Nothing
Set idxNew = Nothing
Exit Sub

```

IndexNullsXError:

```

If Not rstEmployees Is Nothing Then

```

```
        If rstEmployees.State = adStateOpen Then rstEmployees.Close
    End If
    Set rstEmployees = Nothing

    ' Delete new Index because this is a demonstration.
    If Not catNorthwind Is Nothing Then
        catNorthwind.Tables("Employees").Indexes.Delete idxNew.Name
    End If

    If Not cnn Is Nothing Then
        If cnn.State = adStateOpen Then cnn.Close
    End If
    Set cnn = Nothing

    Set catNorthwind = Nothing
    Set idxNew = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If

End Sub
' EndIndexNullsVB
```

See Also

[Index Object](#) | [IndexNulls Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example (VB)

The following code demonstrates how to create a new foreign key. It assumes two tables (**Customers** and **Orders**) exist.

```
' BeginCreateKeyVB
Sub Main()
    On Error GoTo CreateKeyError

    Dim kyForeign As New ADOX.Key
    Dim cat As New ADOX.Catalog

    ' Connect the catalog
    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Define the foreign key
    kyForeign.Name = "CustOrder"
    kyForeign.Type = adKeyForeign
    kyForeign.RelatedTable = "Customers"
    kyForeign.Columns.Append "CustomerId"
    kyForeign.Columns("CustomerId").RelatedColumn = "CustomerId"
    kyForeign.UpdateRule = adRICascade

    ' Append the foreign key
    cat.Tables("Orders").Keys.Append kyForeign

    'Delete the Key as this is a demonstration
    cat.Tables("Orders").Keys.Delete kyForeign.Name

    'Clean up
    Set cat.ActiveConnection = Nothing
    Set cat = Nothing
    Set kyForeign = Nothing
    Exit Sub

CreateKeyError:
    Set cat = Nothing
    Set kyForeign = Nothing
```

```
If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
```

```
End Sub
' EndCreateKeyVB
```

See Also

[Append Method \(Columns\)](#) | [Append Method \(Keys\)](#) | [Catalog Object](#) | [Column Object](#) | [Columns Collection](#) | [Key Object](#) | [Keys Collection](#) | [Name Property](#) | [RelatedColumn Property](#) | [RelatedTable Property](#) | [Table Object](#) | [Tables Collection](#) | [Type Property \(Key\)](#) | [UpdateRule Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

NumericScale and Precision Properties Example (VB)

This example demonstrates the [NumericScale](#) and [Precision](#) properties of the [Column](#) object. This code displays their value for the **Order Details** table of the *Northwind* database.

```
' BeginNumericScalePrecVB
Sub Main()
    On Error GoTo NumericScalePrecXError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim tblOD As ADOX.Table
    Dim colLoop As ADOX.Column

    ' Connect the catalog.
    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "data source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb'";"
    Set cat.ActiveConnection = cnn

    ' Retrieve the Order Details table
    Set tblOD = cat.Tables("Order Details")

    ' Display numeric scale and precision of
    ' small integer fields.
    For Each colLoop In tblOD.Columns
        If colLoop.Type = adSmallInt Then
            MsgBox "Column: " & colLoop.Name & vbCr & _
                "Numeric scale: " & _
                colLoop.NumericScale & vbCr & _
                "Precision: " & colLoop.Precision
        End If
    Next colLoop

    'Clean up
    cnn.Close
    Set cat = Nothing
    Set cnn = Nothing
    Exit Sub

NumericScalePrecXError:
```

```
Set cat = Nothing

If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
' EndNumericScalePrecVB
```

See Also

[Column Object](#) | [NumericScale Property](#) | [Precision Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Parameters Collection, Command Property Example (VB)

The following code demonstrates how to use the [Command](#) property with the [Command](#) object to retrieve parameter information for the procedure.

```
' BeginParametersVB
Sub Main()
    On Error GoTo ProcedureParametersError

    Dim cnn As New ADODB.Connection
    Dim cmd As ADODB.Command
    Dim prm As ADODB.Parameter
    Dim cat As New ADOX.Catalog

    ' Open the Connection
    cnn.Open _
        "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Open the catalog
    Set cat.ActiveConnection = cnn

    ' Get the command object
    Set cmd = cat.Procedures("CustomerById").Command

    ' Retrieve Parameter information
    cmd.Parameters.Refresh
    For Each prm In cmd.Parameters
        Debug.Print prm.Name & ":" & prm.Type
    Next

    'Clean up
    cnn.Close
    Set cat = Nothing
    Set cmd = Nothing
    Set cnn = Nothing
    Exit Sub

ProcedureParametersError:

    Set cat = Nothing
    Set cmd = Nothing
```

```
If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
' EndParametersVB
```

See Also

[ActiveConnection Property](#) | [Catalog Object](#) | [Command Property](#) | [Procedure Object](#) | [Procedures Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

ParentCatalog Property Example (VB)

The following code demonstrates how to use the [ParentCatalog](#) property to access a [provider](#)-specific property prior to appending a table to a catalog. The property is `AutoIncrement`, which creates an `AutoIncrement` field in a Microsoft Jet database.

```
' BeginCreateAutoIncrColumnVB
Sub Main()
    On Error GoTo CreateAutoIncrColumnError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim tbl As New ADOX.Table

    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';"
    Set cat.ActiveConnection = cnn

    With tbl
        .Name = "MyContacts"
        Set .ParentCatalog = cat
        ' Create fields and append them to the new Table object.
        .Columns.Append "ContactId", adInteger
        ' Make the ContactId column and auto incrementing column
        .Columns("ContactId").Properties("AutoIncrement") = True
        .Columns.Append "CustomerID", adVarChar
        .Columns.Append "FirstName", adVarChar
        .Columns.Append "LastName", adVarChar
        .Columns.Append "Phone", adVarChar, 20
        .Columns.Append "Notes", adLongVarChar
    End With

    cat.Tables.Append tbl
    Debug.Print "Table 'MyContacts' is added."

    ' Delete the table as this is a demonstration.
    cat.Tables.Delete tbl.Name
    Debug.Print "Table 'MyContacts' is delete."

    'Clean up
```

```
cnn.Close
Set cat = Nothing
Set tbl = Nothing
Set cnn = Nothing
Exit Sub
```

CreateAutoIncrColumnError:

```
Set cat = Nothing
Set tbl = Nothing
```

```
If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing
```

```
If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
```

```
End Sub
' EndCreateAutoIncrColumnVB
```

See Also

[Append Method \(Columns\)](#) | [Append Method \(Tables\)](#) | [Catalog Object](#) | [Column Object](#) | [Columns Collection](#) | [Name Property](#) | [ParentCatalog Property](#) | [Table Object](#) | [Type Property \(Column\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

PrimaryKey and Unique Properties Example (VB)

This example demonstrates the [PrimaryKey](#) and [Unique](#) properties of an [Index](#). The code creates a new table with two columns. The **PrimaryKey** and **Unique** properties are used to make one column the primary key for which duplicate values are not allowed.

```
' BeginPrimaryKeyVB
Sub Main()
    On Error GoTo PrimaryKeyXError

    Dim catNorthwind As New ADOX.Catalog
    Dim tblNew As New ADOX.Table
    Dim idxNew As New ADOX.Index
    Dim idxLoop As New ADOX.Index
    Dim colLoop As New ADOX.Column

    ' Connect the catalog
    catNorthwind.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';"

    ' Name new table
    tblNew.Name = "NewTable"

    ' Append a numeric and a text field to new table.
    tblNew.Columns.Append "NumField", adInteger, 20
    tblNew.Columns.Append "TextField", adVarChar, 20

    ' Append new Primary Key index on NumField column
    ' to new table
    idxNew.Name = "NumIndex"
    idxNew.Columns.Append "NumField"
    idxNew.PrimaryKey = True
    idxNew.Unique = True
    tblNew.Indexes.Append idxNew

    ' Append an index on Textfield to new table.
    ' Note the different technique: Specifying index and
    ' column name as parameters of the Append method
    tblNew.Indexes.Append "TextIndex", "TextField"
```

```

' Append the new table
catNorthwind.Tables.Append tblNew

With tblNew
    Debug.Print tblNew.Indexes.Count & " Indexes in " & _
        tblNew.Name & " Table"

    ' Enumerate Indexes collection.
    For Each idxLoop In .Indexes
        With idxLoop
            Debug.Print "Index " & .Name
            Debug.Print "    Primary key = " & .PrimaryKey
            Debug.Print "    Unique = " & .Unique

            ' Enumerate Columns collection of each Index
            ' object.
            Debug.Print "    Columns"
            For Each colLoop In .Columns
                Debug.Print "        " & colLoop.Name
            Next colLoop
        End With
    Next idxLoop

End With

' Delete new table as this is a demonstration.
catNorthwind.Tables.Delete tblNew.Name

'Clean up
Set catNorthwind.ActiveConnection = Nothing
Set catNorthwind = Nothing
Set tblNew = Nothing
Set idxNew = Nothing
Set idxLoop = Nothing
Set colLoop = Nothing
Exit Sub

PrimaryKeyXError:

Set catNorthwind = Nothing
Set tblNew = Nothing
Set idxNew = Nothing
Set idxLoop = Nothing
Set colLoop = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
' EndPrimaryKeyVB

```

See Also

[Index Object](#) | [PrimaryKey Property](#) | [Unique Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Procedures Append Method Example (VB)

The following code demonstrates how to use a [Command](#) object and the [Procedures](#) collection [Append](#) method to create a new procedure in the underlying [data source](#).

```
' BeginCreateProcedureVB
Sub Main()
    On Error GoTo CreateProcedureError

    Dim cnn As New ADODB.Connection
    Dim cmd As New ADODB.Command
    Dim prm As ADODB.Parameter
    Dim cat As New ADOX.Catalog

    ' Open the Connection
    cnn.Open _
        "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Create the parameterized command (Microsoft Jet specific)
    Set cmd.ActiveConnection = cnn
    cmd.CommandText = "PARAMETERS [CustId] Text;" & _
        "Select * From Customers Where CustomerId = [CustId]"

    ' Open the Catalog
    Set cat.ActiveConnection = cnn

    ' Create the new Procedure
    cat.Procedures.Append "CustomerById", cmd

    'Clean
    cnn.Close
    Set cat = Nothing
    Set cmd = Nothing
    Set cnn = Nothing
    Exit Sub

CreateProcedureError:
    Set cat = Nothing
    Set cmd = Nothing
```

```
If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
' EndCreateProcedureVB
```

See Also

[ActiveConnection Property](#) | [Append Method \(Procedures\)](#) | [Catalog Object](#) | [Procedure Object](#) | [Procedures Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Procedures Delete Method Example (VB)

The following code demonstrates how to delete a procedure using the [Procedures](#) collection [Delete](#) method.

```
' BeginDeleteProcedureVB
Sub Main()
    On Error GoTo DeleteProcedureError

    Dim cat As New ADOX.Catalog

    ' Open the Catalog.
    cat.ActiveConnection = _
        "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Delete the Procedure.
    cat.Procedures.Delete "CustomerById"

    'Clean up
    Set cat.ActiveConnection = Nothing
    Set cat = Nothing
    Exit Sub

DeleteProcedureError:
    Set cat = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndDeleteProcedureVB
```

See Also

[ActiveConnection Property](#) | [Catalog Object](#) | [Delete Method \(Collections\)](#) | [Procedure Object](#) | [Procedures Collection](#)

ADOX 2.5 

Procedures Refresh Method Example (VB)

The following code shows how to refresh the [Procedures](#) collection of a [Catalog](#). This is required before [Procedure](#) objects from the **Catalog** can be accessed.

```
' BeginProceduresRefreshVB
Sub Main()
    On Error GoTo ProcedureRefreshError

    Dim cat As New ADOX.Catalog

    ' Open the Catalog
    cat.ActiveConnection = _
        "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';"

    ' Refresh the Procedures collection
    cat.Procedures.Refresh

    'Clean up
    Set cat.ActiveConnection = Nothing
    Set cat = Nothing
    Exit Sub

ProcedureRefreshError:
    Set cat = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndProceduresRefreshVB
```

See Also

[Catalog Object](#) | [Procedures Collection](#) | [Refresh Method](#)

ADOX 2.5 

SortOrder Property Example (VB)

This example demonstrates the [SortOrder](#) property of a [Column](#) that has been appended to the [Columns](#) collection of an [Index](#). The code appends an ascending index to the Country column in the **Employees** table, then displays the records. Then the code appends a descending index to the Country column in the **Employees** table and displays the records again. The difference between ascending and descending indexes is shown.

```
' BeginSortOrderVB
Sub Main()
    On Error GoTo SortOrderXError

    Dim cnn As New ADODB.Connection
    Dim catNorthwind As New ADOX.Catalog
    Dim idxAscending As New ADOX.Index
    Dim idxDescending As New ADOX.Index
    Dim rstEmployees As New ADODB.Recordset

    ' Connect the catalog.
    cnn.Open "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';"
    Set catNorthwind.ActiveConnection = cnn

    ' Append Country column to new index
    idxAscending.Columns.Append "Country"
    idxAscending.Columns("Country").SortOrder = adSortAscending
    idxAscending.Name = "Ascending"
    idxAscending.IndexNulls = adIndexNullsAllow

    'Append new index to Employees table
    catNorthwind.Tables("Employees").Indexes.Append idxAscending

    rstEmployees.Index = idxAscending.Name
    rstEmployees.Open "Employees", cnn, adOpenKeyset, _
        adLockOptimistic, adCmdTableDirect

    With rstEmployees
        .MoveFirst
        Debug.Print "Index = " & .Index
        Debug.Print "  Country - Name"

        ' Enumerate the Recordset. The value of the
```

```

' IndexNulls property will determine if the newly
' added record appears in the output.
Do While Not .EOF
    Debug.Print "      " & !Country & " - " & _
        !FirstName & " " & !LastName
    .MoveNext
Loop

.Close
End With

' Append Country column to new index
idxDescending.Columns.Append "Country"
idxDescending.Columns("Country").SortOrder = adSortDescending
idxDescending.Name = "Descending"
idxDescending.IndexNulls = adIndexNullsAllow

'Append descending index to Employees table
catNorthwind.Tables("Employees").Indexes.Append idxDescending

rstEmployees.Index = idxDescending.Name
rstEmployees.Open "Employees", cnn, adOpenKeyset, _
    adLockOptimistic, adCmdTableDirect

With rstEmployees
    .MoveFirst
    Debug.Print "Index = " & .Index
    Debug.Print "  Country - Name"

    ' Enumerate the Recordset. The value of the
    ' IndexNulls property will determine if the newly
    ' added record appears in the output.
    Do While Not .EOF
        Debug.Print "      " & !Country & " - " & _
            !FirstName & " " & !LastName
        .MoveNext
    Loop

    .Close
End With

' Delete new Indexes because this is a demonstration.
catNorthwind.Tables("Employees").Indexes.Delete idxAscending.Nam
catNorthwind.Tables("Employees").Indexes.Delete idxDescending.Na

'Clean up
cnn.Close
Set catNorthwind = Nothing
Set idxAscending = Nothing
Set idxDescending = Nothing

```

```
Set rstEmployees = Nothing
Set cnn = Nothing
Exit Sub
```

SortOrderXError:

```
Set catNorthwind = Nothing
Set idxAscending = Nothing
Set idxDescending = Nothing
```

```
If Not rstEmployees Is Nothing Then
    If rstEmployees.State = adStateOpen Then rstEmployees.Close
End If
Set rstEmployees = Nothing
```

```
If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing
```

```
If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
```

```
End Sub
' EndSortOrderVB
```

See Also

[Column Object](#) | [Columns Collection](#) | [Index Object](#) | [SortOrder Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Views and Fields Collections Example (VB)

The following code demonstrates how to use the [Command](#) property and the [Recordset](#) object to retrieve field information for the view.

```
' BeginViewFieldsVB
Sub ViewFields()
    On Error GoTo ViewFieldsError

    Dim cnn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    Dim fld As ADODB.Field
    Dim cat As New ADOX.Catalog

    ' Open the Connection
    cnn.Open _
        "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Open the catalog
    Set cat.ActiveConnection = cnn

    ' Set the Source for the Recordset
    Set rst.Source = cat.Views("AllCustomers").Command

    ' Retrieve Field information
    rst.Fields.Refresh
    For Each fld In rst.Fields
        Debug.Print fld.Name & ":" & fld.Type
    Next

    'Clean up
    cnn.Close
    Set cat = Nothing
    Set rst = Nothing
    Set cnn = Nothing
    Exit Sub

ViewFieldsError:

    If Not cnn Is Nothing Then
        If cnn.State = adStateOpen Then cnn.Close
```

```
End If

Set cat = Nothing
Set rst = Nothing
Set cnn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If

End Sub
' EndViewFieldsVB
```

See Also

[ActiveConnection Property](#) | [Catalog Object](#) | [Command Property](#) | [View Object](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Views Append Method Example (VB)

The following code demonstrates how to use a [Command](#) object and the [Views](#) collection [Append](#) method to create a new view in the underlying [data source](#).

```
' BeginCreateViewVB
Sub Main()
    On Error GoTo CreateViewError

    Dim cmd As New ADODB.Command
    Dim cat As New ADOX.Catalog

    ' Open the Catalog
    cat.ActiveConnection = _
        "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Create the command representing the view.
    cmd.CommandText = "Select * From Customers"

    ' Create the new View
    cat.Views.Append "AllCustomers", cmd

    'Clean up
    Set cat.ActiveConnection = Nothing
    Set cat = Nothing
    Set cmd = Nothing
    Exit Sub

CreateViewError:

    Set cat = Nothing
    Set cmd = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndCreateViewVB
```

See Also

[ActiveConnection Property](#) | [Append Method \(Views\)](#) | [Catalog Object](#) | [View](#)

[Object](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Views Collection, CommandText Property Example (VB)

The following code demonstrates how to use the [Command](#) property to update the text of a view.

```
' BeginViewsCollectionVB
Sub Main()
    On Error GoTo ViewTextError

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim cmd As New ADODB.Command

    ' Open the Connection
    cnn.Open _
        "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    ' Open the catalog
    Set cat.ActiveConnection = cnn

    ' Get the command
    Set cmd = cat.Views("AllCustomers").Command

    ' Update the CommandText of the Command
    cmd.CommandText = _
        "Select CustomerId, CompanyName, ContactName From Customers"

    ' Update the View
    Set cat.Views("AllCustomers").Command = cmd

    'Clean up
    cnn.Close
    Set cat = Nothing
    Set cmd = Nothing
    Set cnn = Nothing
    Exit Sub

ViewTextError:

    Set cat = Nothing
    Set cmd = Nothing
```

```
If Not cnn Is Nothing Then
    If cnn.State = adStateOpen Then cnn.Close
End If
Set cnn = Nothing

If Err <> 0 Then
    MsgBox Err.Source & "-->" & Err.Description, , "Error"
End If
End Sub
' EndViewsCollectionVB
```

See Also

[ActiveConnection Property](#) | [Catalog Object](#) | [Command Property](#) | [View Object](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Views Delete Method Example (VB)

The following code shows how to use the [Delete](#) method to delete a view from the catalog.

```
' BeginDeleteViewVB
Sub Main()
    On Error GoTo DeleteViewError

    Dim cat As New ADOX.Catalog

    ' Open the catalog
    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb';"

    'Delete the View
    cat.Views.Delete "AllCustomers"

    'Clean up
    Set cat.ActiveConnection = Nothing
    Set cat = Nothing
    Exit Sub

DeleteViewError:
    Set cat = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndDeleteViewVB
```

See Also

[Delete Method \(Collections\)](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Views Refresh Method Example (VB)

The following code shows how to refresh the [Views](#) collection of a [Catalog](#). This is required before [View](#) objects from the **Catalog** can be accessed.

```
' BeginViewsRefreshVB
Sub Main()
    On Error GoTo ProcedureViewsRefreshError

    Dim cat As New ADOX.Catalog

    ' Open the Catalog
    cat.ActiveConnection = "Provider='Microsoft.Jet.OLEDB.4.0';" & _
        "Data Source='c:\Program Files\" & _
        "Microsoft Office\Office\Samples\Northwind.mdb';"

    ' Refresh the Procedures collection
    cat.Views.Refresh

    'Clean up
    Set cat = Nothing
    Exit Sub

ProcedureViewsRefreshError:

    If Not cat Is Nothing Then
        Set cat = Nothing
    End If

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
' EndViewsRefreshVB
```

See Also

[Refresh Method](#) | [Views Collection](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

ADOX Code Examples in Microsoft Visual C++

These topics provide sample code to help you understand how to use ADOX. All code examples are written using Microsoft Visual C++.

Note Paste the entire code example, from beginning to end, into your code editor. The example may not run correctly if you use partial examples or if paragraph formatting is lost.

Methods

- [Columns and Tables Append Methods, Name Property Example \(VC++\)](#)
- [Connection Close Method, Table Type Property Example \(VC++\)](#)
- [Create Method Example \(VC++\)](#)
- [GetObjectOwner and SetObjectOwner Methods Example \(VC++\)](#)
- [GetPermissions and SetPermissions Methods Example \(VC++\)](#)
- [Groups and Users Append, ChangePassword Methods Example \(VC++\)](#)
- [Indexes Append Method Example \(VC++\)](#)
- [Keys Append Method, Key Type, RelatedColumn, RelatedTable, and UpdateRule Properties Example \(VC++\)](#)

Properties

- [Attributes Property Example \(VC++\)](#)
- [Catalog ActiveConnection Property Example \(VC++\)](#)
- [Clustered Property Example \(VC++\)](#)
- [Command and CommandText Properties Example \(VC++\)](#)
- [Command Property, Parameters Collection Example \(VC++\)](#)
- [DateCreated and DateModified Properties Example \(VC++\)](#)
- [DefinedSize Property Example \(VC++\)](#)
- [DeleteRule Property Example \(VC++\)](#)
- [IndexNulls Property Example \(VC++\)](#)
- [Key Type, RelatedColumn, RelatedTable, UpdateRule Properties, Keys Append Method Example \(VC++\)](#)
- [Name Property, Columns and Tables Append Methods Example \(VC++\)](#)

- [NumericScale and Precision Properties Example \(VC++\)](#)
- [ParentCatalog Property Example \(VC++\)](#)
- [PrimaryKey and Unique Properties Example \(VC++\)](#)
- [Table Type Property, Connection Close Method Example \(VC++\)](#)

Collections

- [Parameters Collection, Command Property Example \(VC++\)](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Attributes Property Example (VC++)

This example demonstrates the [Attributes](#) property of a [Column](#). Setting it to **adColNullable** allows the user to set the value of a [Recordset Field](#) to an empty string. In this situation, the user can distinguish between a record where data is not known and a record where the data does not apply.

```
// BeginAttributesCpp
#import "c:\Program Files\Common Files\system\ado\msado15.dll"
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace

#include "iostream.h"
#include "stdio.h"
#include "conio.h"
#include "ADOXAttributesX.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void AttributesX(void);
inline char* mygets(char* strDest, int n)
{
    char strExBuff[10];
    char* pstrRet = fgets(strDest, n, stdin);

    if (pstrRet == NULL)
        return NULL;

    if (!strchr(strDest, '\n'))
        // Exhaust the input buffer.
        do
        {
            fgets(strExBuff, sizeof(strExBuff), stdin);
        }while (!strchr(strExBuff, '\n'));
    else
        // Replace '\n' with '\0'
        strDest[strchr(strDest, '\n') - strDest] = '\0';

    return pstrRet;
}

////////////////////////////////////
//                                     //
//          Main Function              //
//                                     //
//                                     //
```

```

////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    AttributesX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                     //
//      AttributesX Function           //
//                                     //
////////////////////////////////////
void AttributesX(void)
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _CatalogPtr m_pCatalog = NULL;
    _ColumnPtr m_pColumn = NULL;
    _TablePtr m_pTable = NULL;

    // Define ADODB object pointers
    ADODB::_ConnectionPtr m_pCnn = NULL;
    ADODB::_RecordsetPtr m_pRstEmployees = NULL;

    IADORecordBinding *picRs = NULL; // Interface Pointer Declare
    CEmployeeRs emprs; // C++ Class Object

    // Define string variables.
    _bstr_t strcnn("Provider='Microsoft.JET.OLEDB.4.0';"
        "Data Source= 'c:\\Program Files\\Microsoft Office\\"
        "Office\\Samples\\Northwind.mdb'");

    try
    {
        // Connect the catalog.
        TESTHR(hr = m_pCnn.CreateInstance(__uuidof (ADODB::Connectio
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog)));
        TESTHR(hr = m_pColumn.CreateInstance(__uuidof(Column)));
        TESTHR(hr = m_pRstEmployees.CreateInstance(__uuidof(ADODB::R

        m_pCnn->Open(strcnn, "", "", NULL);
        m_pCatalog->PutActiveConnection(
            _variant_t((IDispatch *) m_pCnn));
    }
}

```

```

m_pTable= m_pCatalog->Tables->GetItem("Employees");

// Create a new Field object and append it to the Fields
// collection of the Employees table.
m_pColumn->Name = "FaxPhone";
m_pColumn->Type = adVarChar;
m_pColumn->DefinedSize = 24;
m_pColumn->Attributes = adColNullable;

m_pCatalog->Tables->GetItem("Employees")->Columns->
    Append(m_pColumn->Name, adVarChar, 24);
//Append("FaxPhone", adVarChar, 24);

// Open the Employees table for updating as a Recordset.
m_pRstEmployees->Open("Employees",
    _variant_t((IDispatch *) m_pCnn),
    ADODB::adOpenKeyset, ADODB::adLockOptimistic,
    ADODB::adCmdTable);

// Get user input.
printf("Enter fax number for : %s %s\n", (LPSTR) (_bstr_t)
    m_pRstEmployees->Fields->GetItem("LastName")->Value,
    (LPSTR) (_bstr_t) m_pRstEmployees->Fields->
    GetItem("FirstName")->Value);
printf("[? - unknown, X - has no fax] : \n");
char strInput[10];
mygets(strInput, 10);
char* strTemp = strtok(strInput, " \t");
_variant_t vNull;
vNull.vt = VT_BSTR;
vNull.bstrVal = NULL;
if(strTemp!=NULL)
{
    if(strcmp(strTemp, "?") == 0)
    {
        m_pRstEmployees->Fields->GetItem("FaxPhone")->
            PutValue(vNull);
    }
    else if( (strcmp(strTemp, "X") == 0) | (strcmp(strTemp, "x"
    {
        m_pRstEmployees->Fields->GetItem("FaxPhone")->
            PutValue("");
    }
    else
    {
        m_pRstEmployees->Fields->GetItem("FaxPhone")->
            PutValue(strTemp);
    }
    m_pRstEmployees->Update();
}

```

```

// Open an IADORecordBinding interface pointer which
// we will use for binding Recordset to a class
TESTHR(hr = m_pRstEmployees->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

// Bind the Recordset to a C++ class here
TESTHR(hr = picRs->BindToRecordset(&emprs));

// Print report.
printf("\nName - Fax number\n");
printf("%s %s ", emprs.lemp_LastNameStatus == adFldOK ?
    emprs.m_szemp_LastName : "<NULL>",
    emprs.lemp_FirstNameStatus == adFldOK ?
    emprs.m_szemp_FirstName : "<NULL>");

if (emprs.lemp_FaxphoneStatus == adFldNull)
    printf("- [Unknown]\n");
else if (strcmp((LPSTR)emprs.m_szemp_Faxphone, "") == 0)
    printf("- [Has no fax]\n");
else
    printf("- %s\n", emprs.m_szemp_Faxphone);

}

// Delete new field because this is a demonstration.
//m_pTable->Columns->Delete(m_pColumn->Name);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in AttributesX...."<< endl;
}

if (m_pRstEmployees)
    if (m_pRstEmployees->State == 1)
        m_pRstEmployees->Close();

// Delete new field because this is a demonstration.
if (m_pTable != NULL)
    m_pTable->Columns->Delete(m_pColumn->Name);

```

```

    if (m_pCnn)
        if (m_pCnn->State == 1)
            m_pCnn->Close();

    // Release the IADORecordset Interface here
    if(picRs)
        picRs->Release();
}
// EndAttributesCpp

```

ADOXAttributesX.h

```

// BeginAttributesH
#include "icrsint.h"

//This class extracts LastName, FirstName, FaxPhone from Employees t

class CEmployeeRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)

    // Column LastName is the 2nd field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(2,adVarChar,m_szemp_LastName,
        sizeof(m_szemp_LastName),lemp_LastNameStatus,TRUE)

    // Column FirstName is the 17th field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(17,adVarChar,m_szemp_FirstName,
        sizeof(m_szemp_FirstName),lemp_FirstNameStatus,TRUE)

    // Column FaxPhone is the 18th field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(18,adVarChar,m_szemp_Faxphone,
        sizeof(m_szemp_Faxphone),lemp_FaxphoneStatus,TRUE)

END_ADO_BINDING()

public:
    CHAR m_szemp_LastName[21];
    ULONG lemp_LastNameStatus;
    CHAR m_szemp_FirstName[11];
    ULONG lemp_FirstNameStatus;
    CHAR m_szemp_Faxphone[25];
    ULONG lemp_FaxphoneStatus;
};
// EndAttributesH

```

See Also

[Attributes Property](#) | [Column Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Catalog ActiveConnection Property Example (VC++)

Setting the [ActiveConnection](#) property to a valid, open connection "opens" the catalog. From an open catalog, you can access the schema objects contained within that catalog.

```
// BeginActiveConnectionCpp
#import "c:\Program Files\Common Files\system\ado\msado15.dll"
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void OpenConnectionX(void);
void OpenConnectionWithStringX(void);

////////////////////////////////////
//                               //
//      Main Function            //
//                               //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    OpenConnectionX();

    OpenConnectionWithStringX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//      OpenConnectionX Function //
//                               //
////////////////////////////////////
```

```

void OpenConnectionX(void)
{
    HRESULT    hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _CatalogPtr m_pCatalog = NULL;

    //Define ADODB object pointers
    ADODB::_ConnectionPtr m_pCnn    = NULL;

    // Define string variables.
    _bstr_t strcnn("Provider='Microsoft.JET.OLEDB.4.0';"
        "Data source = 'c:\\Program Files\\Microsoft Office\\"
        "Office\\Samples\\Northwind.mdb'");

    try
    {
        TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog)));
        m_pCnn->Open(strcnn, "", "", NULL);
        m_pCatalog->PutActiveConnection(_variant_t((IDispatch *) m_p
        _variant_t vIndex = (short) 0;
        cout<<m_pCatalog->Tables->GetItem(vIndex)->Type<<endl;
    }
    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());

        printf("\n\tSource :  %s \n\tdescription : %s \n ",
            (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
    }
    catch(...)
    {
        cout << "Error ocured in OpenConnectionX...."<< endl;
    }

    if (m_pCnn)
        if (m_pCnn->State == 1)
            m_pCnn->Close();
}

////////////////////////////////////
//                                     //
//          OpenConnectionWithStringX Function          //
//                                     //
////////////////////////////////////

```

```

void OpenConnectionWithStringX(void)
{
    HRESULT    hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _CatalogPtr m_pCatalog = NULL;

    // Define string variables.
    _bstr_t strcnn("Provider='Microsoft.JET.OLEDB.4.0';"
        "Data source = 'c:\\Program Files\\Microsoft Office\\"
        "Office\\Samples\\Northwind.mdb'");

    try
    {
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog));
        m_pCatalog->PutActiveConnection(strcnn);
        _variant_t vIndex = (short) 0;
        cout<<m_pCatalog->Tables->GetItem(vIndex)->Type<<endl;
    }
    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());

        printf("\n\tSource : %s \n\tdescription : %s \n ",
            (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
    }
    catch(...)
    {
        cout << "Error occured in OpenConnectionWithStringX...."<< e
    }
}
// EndActiveConnectionCpp

```

See Also

[ActiveConnection Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Clustered Property Example (VC++)

This example demonstrates the [Clustered](#) property of an [Index](#). Note that Microsoft Jet databases do not support clustered indexes, so this example will return **False** for the **Clustered** property of all indexes in the *Northwind* database.

```
// BeginClusteredCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void ClusteredX(void);

////////////////////////////////////
//                               //
//      Main Function            //
//                               //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    ClusteredX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//      ClusteredX Function     //
//                               //
////////////////////////////////////
void ClusteredX()
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
```

```

_CatalogPtr m_pCatalog = NULL;
_TablePtr m_pTable = NULL;
_IndexPtr m_pIndex = NULL;

//Define other variables here
_variant_t vIndex;
try
{
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));

    // Connect to the catalog.
    m_pCatalog->PutActiveConnection(
        "Provider='Microsoft.JET.OLEDB.4.0';data source="
        "'c:\\Program Files\\Microsoft Office\\Office\\Samples"
        "\\Northwind.mdb';");

    int iLineCnt = 1;
    //Enumerate Tables.
    for(short iTable = 0;iTable < m_pCatalog->Tables->Count;iTab
    {
        vIndex = iTable;
        m_pTable = m_pCatalog->Tables->GetItem(vIndex);

        //Enumerate Indexes.
        for(short iIndex = 0;iIndex < m_pTable->Indexes->Count;
            iIndex++)
        {
            vIndex = iIndex;
            m_pIndex = m_pTable->Indexes->GetItem(vIndex);
            cout << m_pTable->Name << "    " ;
            cout << m_pIndex->Name << "    " << (m_pIndex->
                GetClustered() ? "True" : "False") << endl;

            if (iLineCnt%15 == 0)
            {
                printf("\nPress any key to continue...\n");
                getch();
            }
            iLineCnt++;
        }
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}

```

```
    }  
    catch(...)  
    {  
        cout << "Error occured in ClusteredX...."<< endl;  
    }  
}  
// EndClusteredCpp
```

See Also

[Clustered Property](#) | [Index Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Columns and Tables Append Methods, Name Property Example (VC++)

The following code demonstrates how to create a new table.

```
// BeginCreateTableCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void CreateTableX(void);

////////////////////////////////////
//                               //
//      Main Function             //
//                               //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    CreateTableX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//      CreateTableX Function    //
//                               //
////////////////////////////////////
void CreateTableX()
{
    HRESULT hr = S_OK;
```

```

// Define ADOX object pointers.
// Initialize pointers on define.
// These are in the ADOX:: namespace.
_CatalogPtr m_pCatalog = NULL;
_TablePtr m_pTable = NULL;

try
{
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog))

    //Open the catalog
    m_pCatalog->PutActiveConnection(
        "Provider='Microsoft.JET.OLEDB.4.0';" \
        "data source='c:\\Program Files\\Microsoft Office"
        "\\Office\\Samples\\Northwind.mdb';");

    TESTHR(hr = m_pTable.CreateInstance(__uuidof(Table));
    m_pTable->PutName("MyTable");
    m_pTable->Columns->Append("Column1", adInteger, 0);
    m_pTable->Columns->Append("Column2", adInteger, 0);
    m_pTable->Columns->Append("Column3", adVarChar, 50);
    m_pCatalog->Tables->Append(
        _variant_t((IDispatch *)m_pTable));
    printf("Table 'MyTable' is added.\n");

    // Delete the table as this is a demonstration.
    m_pCatalog->Tables->Delete("MyTable");
    printf("Table 'MyTable' is deleted.\n");
}

catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}

catch(...)
{
    cout << "Error occured in include files...."<< endl;
}
}
// EndCreateTableCpp

```

See Also

[Append Method \(Columns\)](#) | [Append Method \(Tables\)](#) | [Name Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 


```

{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _CatalogPtr m_pCatalog = NULL;

    // Define ADODB object pointers.
    ADODB::_ConnectionPtr m_pCnn = NULL;
    ADODB::_CommandPtr m_pCommand = NULL;

    try
    {
        //Open the Connection
        TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection));
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));
        TESTHR(hr = m_pCommand.CreateInstance(__uuidof(ADODB::Command));
        m_pCnn->Open("Provider='Microsoft.JET.OLEDB.4.0';"
            "data source='c:\\Program Files\\Microsoft Office"
            "\\Office\\Samples\\Northwind.mdb';", "", "", NULL);

        // Open the catalog
        m_pCatalog->PutActiveConnection(_variant_t((IDispatch *)m_pCnn));

        // Get the Command
        m_pCommand = m_pCatalog->Procedures->GetItem("CustomerById");

        // Update the CommandText
        m_pCommand->PutCommandText("PARAMETERS [CustId] Text;select
            \"CustomerId, CompanyName, ContactName \"
            \"from Customers where CustomerId = [CustId]");
        printf("CommandText is updated.\n");

        // Update the Procedure
        m_pCatalog->Procedures->GetItem("CustomerById")->PutCommand(
            _variant_t((IDispatch *)m_pCommand));
        printf("Procedure 'CustomerById' is updated.\n");
    }
    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());

        printf("\n\tSource : %s \n\tdescription : %s \n ", (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
    }
    catch(...)
    {
        cout << "Error occurred in ProcedureTextX..."<< endl;
    }
}

```

```
    }  
}  
// EndCommandTextCpp
```

See Also

[Command Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Connection Close Method, Table Type Property Example (VC++)

Setting the [ActiveConnection](#) property to **Nothing** should "close" the catalog. Associated collections will be empty. Any objects that were created from schema objects in the catalog will be orphaned. Any properties on those objects that have been cached will still be available, but attempting to read properties that require a call to the provider will fail.

```
// BeginCloseConnectionCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace
#import "c:\Program Files\Common Files\system\ado\msado15.dll"

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void CloseConnectionByNothingX(void);
void CloseConnectionX(void);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    CloseConnectionByNothingX();
    CloseConnectionX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//      CloseConnectionByNothingX Function                            //
//                                                                    //
////////////////////////////////////
```

```

////////////////////////////////////
void CloseConnectionByNothingX(void)
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.

    _CatalogPtr m_pCatalog = NULL;
    _TablePtr m_pTable = NULL;

    //Define ADODB object pointers
    ADODB::_ConnectionPtr m_pCnn = NULL;

    //Define other variables
    _variant_t vIndex = (short) 0;

    try
    {
        TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection));
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));

        m_pCnn->Open("Provider='Microsoft.JET.OLEDB.4.0';"
            "Data Source= 'c:\\Program Files\\Microsoft Office\\"
            "Office\\Samples\\Northwind.mdb';", "", "", NULL);

        m_pCatalog->PutActiveConnection(_variant_t((IDispatch *)m_pC
        m_pTable = m_pCatalog->Tables->GetItem(vIndex);

        // Cache m_pTable.Type info
        cout << m_pTable->Type << endl;

        _variant_t vCnn;
        vCnn.vt = VT_DISPATCH;
        vCnn.pdispVal = NULL;
        m_pCatalog->PutActiveConnection(vCnn);

        // m_pTable is orphaned
        cout << m_pTable->Type << endl;
        // Previous line will succeed if this was cached

        cout << m_pTable->Columns->GetItem(vIndex)->DefinedSize << e
        // Previous line will fail if this info has not been cached
    }
    catch(_com_error &e)
    {
        // Notify the user of errors if any.
    }
}

```

```

        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());

        printf("\nError\n\tSource : %s \n\tdescription : %s \n",
            (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
    }
    catch(...)
    {
        cout << "Error occured in CloseConnectionByNothingX...."<< e
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                 //
//      CloseConnectionX Function                                //
//                                                                 //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void CloseConnectionX()
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.

    _CatalogPtr m_pCatalog = NULL;
    _TablePtr m_pTable = NULL;

    //Define ADODB object pointers
    ADODB::_ConnectionPtr m_pCnn = NULL;

    //Define other variables
    _variant_t vIndex = (short) 0;
    try
    {
        TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection
        m_pCnn->Open("Provider='Microsoft.JET.OLEDB.4.0';"
            "Data Source= 'c:\\Program Files\\Microsoft Office\\"
            "Office\\Samples\\Northwind.mdb';", "", "", NULL));

        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));
        m_pCatalog->PutActiveConnection(_variant_t((IDispatch *)m_pC

        m_pTable = m_pCatalog->Tables->GetItem(vIndex);

        // Cache m_pTable.Type info
        cout << m_pTable->Type << endl;

        m_pCnn->Close();
    }
}

```

```

    // m_pTable is orphaned
    cout << m_pTable->Type << endl;
    // Previous line will succeed if this was cached

    cout << m_pTable->Columns->GetItem(vIndex)->DefinedSize << e
    // Previous line will fail if this info has not been cached
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\nError\n\tSource : %s \n\tdescription : %s \n",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in CloseConnectionX...."<< endl;
}
}
// EndCloseConnectionCpp

```

See Also

[ActiveConnection Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Create Method Example (VC++)

The following code shows how to create a new Microsoft Jet database with the [Create](#) method.

```
// BeginCreateDatabaseCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" no_name
#import "c:\Program Files\Common Files\system\ado\msado15.dll"

#define TESTHR(x) if FAILED(x) _com_issue_error(x);

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
void CreateDatabaseX(void);

//-----//
//Main Function
//Purpose:  Test Driver
//-----//
void main()
{
    HRESULT hr = S_OK;

    hr = ::CoInitialize(NULL);
    if(SUCCEEDED(hr))
    {
        CreateDatabaseX();

        //Wait here for the user to see the output
        printf("Press any key to continue...");
        getch();

        ::CoUninitialize();
    }
}

//-----//
//CreateDatabaseX
//Purpose:  create a new Jet database with the Create method
//-----//
void CreateDatabaseX()
{
```

```

HRESULT hr = S_OK;

// Define ADOX object pointers.
// Initialize pointers on define.
// These are in the ADOX:: namespace.

_CatalogPtr m_pCatalog = NULL;

//Set ActiveConnection of Catalog to this string
_bstr_t strcnn("Provider='Microsoft.JET.OLEDB.4.0';"
              "Data source = c:\\new.mdb");
try
{
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog)));
    m_pCatalog->Create(strcnn);
    printf("Database 'c:\\new.mdb' is created.\n");
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
          (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in CreateDatabaseX...."<< endl;
}
}
// EndCreateDatabaseCpp

```

See Also

[Create Method](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 


```

HRESULT hr = S_OK;

// Define ADOX object pointers.
// Initialize pointers on define.
// These are in the ADOX:: namespace.
_CatalogPtr m_pCatalog = NULL;
_TablePtr m_pTblEmployees = NULL;
_TablePtr m_pTblNew = NULL;

//Set ActiveConnection of Catalog to this string
_bstr_t strCnn("Provider='Microsoft.JET.OLEDB.4.0';"
    "Data Source= 'c:\\Program Files\\Microsoft Office\\"
    "Office\\Samples\\Northwind.mdb'");

try
{
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog)));

    // Connect the catalog.
    m_pCatalog->PutActiveConnection(strCnn);

    m_pTblEmployees = m_pCatalog->Tables->GetItem("Employees");

    // Print current information about the Employees table.
    DateOutPut((_bstr_t)"Current properties", m_pTblEmployees);

    // Create and append column to the Employees table.
    m_pTblEmployees->Columns->Append("NewColumn", adInteger,0);

    m_pCatalog->Tables->Refresh();

    // Print new information about the Employees table.
    DateOutPut((_bstr_t)"After creating a new column",
        m_pTblEmployees);

    // Delete new column because this is a demonstration.
    m_pTblEmployees->Columns->Delete("NewColumn");

    // Create and append new Table object to the Northwind datab
    TESTHR(hr = m_pTblNew.CreateInstance(__uuidof(Table)));

    m_pTblNew->Name = "NewTable";
    m_pTblNew->Columns->Append("NewColumn", adInteger,0);
    m_pCatalog->Tables->Append(_variant_t((IDispatch*)m_pTblNew)
    m_pCatalog->Tables->Refresh());

    //Print information about the new Table object.
    DateOutPut((_bstr_t)"After creating a new table", m_pCatalog
        Tables->GetItem("NewTable"));
}

```

```

        // Delete new Table object because this is a demonstration.
        m_pCatalog->Tables->Delete(m_pTblNew->Name);
    }

    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());

        printf("\n\tSource : %s \n\tDescription : %s \n ",
            (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
    }

    catch(...)
    {
        cout << "Error occurred in include files...."<< endl;
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//          DateOutPut Function                                                                 //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void DateOutPut(_bstr_t strTemp , _TablePtr tblTemp)
{
    // Print DateCreated and DateModified information about
    // specified Table object.
    cout << strTemp << endl;
    cout << "      Table: " << tblTemp->GetName() << endl;
    cout << "          DateCreated = " << (_bstr_t)tblTemp->
        GetDateCreated() << endl;
    cout << "          DateModified = " << (_bstr_t)tblTemp->
        GetDateModified() << endl;
}
// EndDateCreatedCpp

```

See Also

[Column Object](#) | [DateCreated Property](#) | [DateModified Property](#) | [Table Object](#)

© 1998-2003 Microsoft Corporation. All rights reserved.

ADOX 2.5 

DefinedSize Property Example (VC++)

This example demonstrates the [DefinedSize](#) property of a [Column](#). The code will redefine the size of the FirstName column of the **Employees** table of the *Northwind* database. Then, the change in the values of the FirstName [Field](#) of a [Recordset](#) based on the **Employees** table is displayed. Note that by default, the FirstName field becomes padded with spaces after you redefine the **DefinedSize** property.

```
// BeginDefinedSizeCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    rename("EOF", "EndOfFile")

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void DefinedSizeX(void);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    DefinedSizeX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                                                    //
//      DefinedSizeX Function                                        //
//                                                                    //
////////////////////////////////////
```

```

//                                                                    //
////////////////////////////////////
void DefinedSizeX(void)
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _CatalogPtr m_pCatNorthwind    = NULL;
    _ColumnPtr m_pColFirstName     = NULL;
    _ColumnPtr m_pColNewFirstName  = NULL;

    // Define ADODB object pointers
    ADODB::_RecordsetPtr m_pRstEmployees = NULL;

    // Define string variables.
    _bstr_t strCnn("Provider='Microsoft.JET.OLEDB.4.0';data source="
        "'c:\\Program Files\\Microsoft Office\\Office\\Samples\\"
        "Northwind.mdb'");
    _bstr_t aryFirstName[10];

    try
    {
        // Open a Recordset for the Employees table.
        TESTHR(hr = m_pRstEmployees.CreateInstance(
            __uuidof(ADODB::Recordset)));
        TESTHR(hr = m_pCatNorthwind.CreateInstance(__uuidof(Catalog));
        TESTHR(hr = m_pColNewFirstName.CreateInstance(__uuidof(Column));

        m_pRstEmployees->Open("Employees", strCnn, ADODB::adOpenKeyset,
            ADODB::adLockReadOnly, ADODB::adCmdTable);

        long lngSize = m_pRstEmployees->RecordCount;
        aryFirstName[lngSize];

        // Open a catalog for the Northwind database,
        // using same connection as rstEmployees.
        m_pCatNorthwind->PutActiveConnection(m_pRstEmployees->
            GetActiveConnection());

        // Loop through the recordset displaying the contents,
        // of the FirstName field, the field's defined size,
        // and its actual size.
        // Also store FirstName values in aryFirstName array.
        m_pRstEmployees->MoveFirst();
        printf("\nOriginal Defined Size and Actual Size");
        int iCount=0;
        while (!(m_pRstEmployees->EndOfFile))
        {

```

```

printf("\nEmployee Name:");
printf("%s ",(LPSTR)(_bstr_t)m_pRstEmployees->Fields->
    GetItem("FirstName")->Value);
printf("%s\n", (LPSTR)(_bstr_t)m_pRstEmployees->Fields->
    GetItem("LastName")->Value);
printf("  FirstName Defined size: %d\n",m_pRstEmployees-
    Fields->GetItem("FirstName")->DefinedSize) ;
printf("  FirstName Actual size: %d\n",m_pRstEmployees->
    Fields->GetItem("FirstName")->ActualSize);
aryFirstName[iCount] = (_bstr_t) m_pRstEmployees->Fields
    GetItem("FirstName")->Value;
m_pRstEmployees->MoveNext();
iCount++;
if(iCount%5==0)
{
    printf("Press any key to continue...");
    getch();
    system("cls");
}
}
m_pRstEmployees->Close();

// Redefine the DefinedSize of FirstName in the catalog.
m_pColFirstName = m_pCatNorthwind->Tables->GetItem("Employee
    Columns->GetItem("FirstName");
m_pColNewFirstName->Name = m_pColFirstName->Name;
m_pColNewFirstName->Type = m_pColFirstName->Type;
m_pColNewFirstName->DefinedSize =
    (m_pColFirstName->DefinedSize) + 1;

// Append new FirstName column to catalog.
m_pCatNorthwind->Tables->GetItem("Employees")->Columns->
    Delete(m_pColFirstName->Name);
m_pCatNorthwind->Tables->GetItem("Employees")->Columns->
    Append(_variant_t((IDispatch*)m_pColNewFirstName,true),
        adVarWChar,m_pColNewFirstName->DefinedSize);

// Open Employee table in Recordset for updating.
m_pRstEmployees->Open("Employees",m_pCatNorthwind->
    GetActiveConnection(),ADODB::adOpenKeyset,
    ADODB::adLockOptimistic,ADODB::adCmdTable);

// Loop through the recordset displaying the contents
// of the FirstName field,the field's defined size,
// and its actual size.
// Also restore FirstName values from aryFirstName.
printf("Press any key to continue...");
getch();
system("cls");

```

```

m_pRstEmployees->MoveFirst();
printf("\n\nNew Defined Size and Actual Size");
iCount=0;
while (!(m_pRstEmployees->EndOfFile))
{
    m_pRstEmployees->Fields->GetItem("FirstName")->Value =
        aryFirstName[iCount];
    printf("\nEmployee Name: ");
    printf("%s ",(LPSTR) (_bstr_t)m_pRstEmployees->Fields->
        GetItem("FirstName")->Value);
    printf("%s\n", (LPSTR)( _bstr_t)m_pRstEmployees->Fields->
        GetItem("LastName")->Value);
    printf("  FirstName Defined size: %d\n",m_pRstEmployees-
        Fields->GetItem("FirstName")->DefinedSize );
    printf("  FirstName Actual size: %d\n",m_pRstEmployees->
        Fields->GetItem("FirstName")->ActualSize );
    m_pRstEmployees->MoveNext();
    iCount++;
    if(iCount%5==0)
    {
        printf("Press any key to continue...");
        getch();
        system("cls");
    }
}
m_pRstEmployees->Close();

// Restore original FirstName column to catalog
m_pCatNorthwind->Tables->GetItem("Employees")->Columns->
    Delete(m_pColNewFirstName->Name);

m_pCatNorthwind->Tables->GetItem("Employees")->Columns->
    Append(_variant_t((IDispatch*)m_pColFirstName,true),
        adVarWChar,m_pColFirstName->DefinedSize);

// Restore original FirstName values to Employees table.
m_pRstEmployees->Open("Employees",m_pCatNorthwind->
    GetActiveConnection(),ADODB::adOpenKeyset,
    ADODB::adLockOptimistic,ADODB::adCmdTable);

m_pRstEmployees->MoveFirst();
iCount = 0;
while(!(m_pRstEmployees->EndOfFile))
{
    m_pRstEmployees->Fields->GetItem("FirstName")->Value =
        aryFirstName[iCount];
    m_pRstEmployees->MoveNext();
    iCount++;
}
}

```

```

catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in DefinedSizeX...."<< endl;
}

if (m_pRstEmployees)
    if (m_pRstEmployees->State == 1)
        m_pRstEmployees->Close();

m_pCatNorthwind = NULL;
}
// EndDefinedSizeCpp

```

See Also

[Column Object](#) | [DefinedSize Property](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

DeleteRule Property Example (VC++)

This example demonstrates the [DeleteRule](#) property of a [Key](#) object. The code appends a new [Table](#) and then defines a new primary key, setting **DeleteRule** to **adRiCascade**.

```
// BeginDeleteRuleCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace
#import "c:\Program Files\Common Files\system\ado\msado15.dll"

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void DeleteRuleX(void);

//////////////////////////////////////
//                                     //
//      Main Function                   //
//                                     //
//////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    DeleteRuleX();

    ::CoUninitialize();
}

//////////////////////////////////////
//                                     //
//      DeleteRuleX Function           //
//                                     //
//////////////////////////////////////
void DeleteRuleX(void)
{
    HRESULT hr = S_OK;
```

```

// Define ADOX object pointers.
// Initialize pointers on define.
// These are in the ADOX:: namespace.
_KeyPtr m_pKeyPrimary = NULL;
_CatalogPtr m_pCatalog = NULL;
_TablePtr m_pTblNew = NULL;

// Define string variables.
_bstr_t strcnn("Provider='Microsoft.JET.OLEDB.4.0';"
              "Data source = 'c:\\Program Files\\Microsoft Office"
              "\\Office\\Samples\\Northwind.mdb';");

try
{
    TESTHR(hr = m_pKeyPrimary.CreateInstance(__uuidof(Key)));
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));
    TESTHR(hr = m_pTblNew.CreateInstance(__uuidof(Table)));

    // Connect the catalog.
    m_pCatalog->PutActiveConnection(strcnn);

    // Name new table.
    m_pTblNew->Name = "NewTable";

    // Append a numeric and a text field to new table.
    m_pTblNew->Columns->Append("NumField",adInteger,20);
    m_pTblNew->Columns->Append("TextField",adVarChar,20);

    // Append the new table.
    m_pCatalog->Tables->Append(_variant_t((IDispatch*)m_pTblNew)
    printf("Table 'NewTable' is added.\n");

    // Define the Primary key.
    m_pKeyPrimary->Name = "NumField";
    m_pKeyPrimary->Type = adKeyPrimary;
    m_pKeyPrimary->RelatedTable = "Customers";
    m_pKeyPrimary->Columns->Append("NumField",adInteger,20);
    m_pKeyPrimary->Columns->GetItem("NumField")->RelatedColumn =
        "CustomerId";
    m_pKeyPrimary->DeleteRule = adRiCascade;

    //to pass an optional column parameter to Key's Append meth
    _variant_t vOptional;
    vOptional.vt = VT_ERROR;
    vOptional.scode = DISP_E_PARAMNOTFOUND;

    // Append the primary key.
    m_pCatalog->Tables->GetItem("NewTable")->Keys->Append(
        _variant_t((IDispatch*)m_pKeyPrimary),
        adKeyPrimary,vOptional,L"",L"");
}

```

```

        // Delete the table as this is a demonstration.
        m_pCatalog->Tables->Delete(m_pTblNew->Name);
        printf("Table 'NewTable' is deleted.\n");
    }

    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());

        printf("\n\tSource : %s \n\tdescription : %s \n ",
            (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
    }

    catch(...)
    {
        cout << "Error occured in include files...."<< endl;
    }
}
// EndDeleteRuleCpp

```

See Also

[DeleteRule Property](#) | [Key Object](#)

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

GetObjectOwner and SetObjectOwner Methods Example (VC++)

This example demonstrates the [GetObjectOwner](#) and [SetObjectOwner](#) methods. This code assumes the existence of the group Accounting (see the [Groups and Users Append, ChangePassword Methods Example \(VC++\)](#) to see how to add this group to the system). The owner of the Categories table is set to Accounting.

```
// BeginOwnersCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" no_name

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void OwnersX(void);

////////////////////////////////////
//                               //
//      Main Function            //
//                               //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    OwnersX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//      OwnersX Function        //
//                               //
////////////////////////////////////
void OwnersX()
```

```

{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _TablePtr m_pTable = NULL;
    _CatalogPtr m_pCatalog = NULL;

    try
    {
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));
        TESTHR(hr = m_pTable.CreateInstance(__uuidof(Table)));

        //Open the Catalog.
        m_pCatalog->PutActiveConnection(
            "Provider='Microsoft.JET.OLEDB.4.0';" \
            "data source='c:\\Program Files\\Microsoft Office\\"
            "Office\\Samples\\Northwind.mdb';"
            "jet oledb:system database='c:\\Program Files\\Microsoft
            "Office\\system.mdw'");

        //Print the original owner of Categories
        _bstr_t strOwner = m_pCatalog->GetObjectOwner("Categories",
            adPermObjTable);
        cout << "Owner of Categories: " << strOwner << "\n" << endl;
        int iLineCnt = 2;

        //Create and append new group with a string.
        m_pCatalog->Groups->Append("Accounting");

        //Set the owner of Categories to Accounting.
        m_pCatalog->SetObjectOwner("Categories",
            adPermObjTable, "Accounting");

        _variant_t vIndex;
        //List the owners of all tables and columns in the catalog.
        for (long iIndex = 0; iIndex < m_pCatalog->Tables->Count; iI
        {
            vIndex = iIndex;
            m_pTable = m_pCatalog->Tables->GetItem(vIndex);
            cout << "Table: " << m_pTable->Name << endl;
            cout << "    Owner: " << m_pCatalog->
                GetObjectOwner(m_pTable->Name, adPermObjTable) << endl;
            if (iLineCnt%16 == 0)
            {
                printf("\nPress any key to continue...\n");
                getch();
            }
            iLineCnt = iLineCnt + 2;
        }
    }
}

```

```

    }

    //Restore the original owner of Categories
    m_pCatalog->SetObjectOwner("Categories",adPermObjTable,strOw

    //Delete Accounting
    m_pCatalog->Groups->Delete("Accounting");
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource,(LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in include files...."<< endl;
}
}
// EndOwnersCpp

```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

GetPermissions and SetPermissions Methods Example (VC++)

This example demonstrates the [GetPermissions](#) and [SetPermissions](#) methods. The following code gives full access to the Orders table to the Admin user.

```
// BeginGrantPermissionCpp
#import "c:\Program Files\Common Files\system\ado\msado15.dll"
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void GrantPermissionsX(void);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Main Function                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    GrantPermissionsX();

    ::CoUninitialize();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                    //
//      GrantPermissionsX Function                                       //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void GrantPermissionsX()
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
```

```

// Initialize pointers on define.
// These are in the ADOX:: namespace.
_CatalogPtr m_pCatalog = NULL;

//Define ADODB object pointers;
ADODB::_ConnectionPtr m_pCnn = NULL;

//Define other variables here.
try
{
    TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection

//Opens a connection to the northwind database
//using the Microsoft Jet 4.0 provider
m_pCnn->PutProvider("Microsoft.Jet.OLEDB.4.0");
m_pCnn->Open("data source='c:\\Program Files\\" \
"Microsoft Office\\Office\\Samples\\Northwind.mdb';" \
"jet oledb:system database='c:\\Program Files\\Microsoft Off
"", "", NULL);

    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));

    m_pCatalog->PutActiveConnection(_variant_t((IDispatch *)m_pC

//Retrieve original permissions
long lngPerm = m_pCatalog->Users->GetItem("admin")->
    GetPermissions("Orders", adPermObjTable);
long lngOrgPerm = lngPerm;
cout << "Original Permissions: " << lngPerm << "\n" << endl;

//Revoke all permissions
m_pCatalog->Users->GetItem("admin")->SetPermissions("Orders"
    adPermObjTable, adAccessRevoke, adRightFull, adInheritNone)

//Display permissions
lngPerm = m_pCatalog->Users->GetItem("admin")->
    GetPermissions("Orders", adPermObjTable);
cout << "Revoked permissions: " << lngPerm << "\n" << endl;

//Give the Admin user full rights on the orders object
m_pCatalog->Users->GetItem("admin")->SetPermissions("Orders"
    adPermObjTable, adAccessSet, adRightFull, adInheritNone);

//Display permissions
lngPerm = m_pCatalog->Users->GetItem("admin")->
    GetPermissions("Orders", adPermObjTable);
cout << "Full permissions: " << lngPerm << "\n" << endl;

//Restore original permissions
m_pCatalog->Users->GetItem("admin")->SetPermissions("Orders"

```

```

        adPermObjTable, adAccessSet, (RightsEnum) lngOrgPerm,
        adInheritNone);

    //Display permissions
    lngPerm = m_pCatalog->Users->GetItem("admin")->
        GetPermissions("Orders", adPermObjTable);
    cout << "Final permissions: " << lngPerm << "\n" << endl;
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in GrantPermissionsX...."<< endl;
}
}
// EndGrantPermissionCpp

```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 


```

void GroupX(void)
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _CatalogPtr m_pCatalog = NULL;
    _UserPtr m_pUserNew = NULL;
    _UserPtr m_pUser = NULL;
    _GroupPtr m_pGroup = NULL;

    // Define String Variables.
    _bstr_t strCnn("Provider='Microsoft.JET.OLEDB.4.0';"
        "Data source = 'c:\\Program Files\\Microsoft Office\\"
        "Office\\Samples\\Northwind.mdb';"
        "jet oledb:system database='c:\\Program Files\\Microsoft Off"
        "Office\\system.mdw'");

    try
    {
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog)));
        m_pCatalog->PutActiveConnection(strCnn);

        // Create and append new group with a string.
        m_pCatalog->Groups->Append("Accounting");

        // Create and append new user with an object.
        TESTHR(hr = m_pUserNew.CreateInstance(__uuidof(User)));
        m_pUserNew->PutName("Pat Smith");
        m_pUserNew->ChangePassword("", "Password1");
        m_pCatalog->Users->Append(
            _variant_t((IDispatch *)m_pUserNew, ""));

        // Make the user Pat Smith a member of the
        // Accounting group by creating and adding the
        // appropriate Group object to the user's Groups
        // collection.The same is accomplished if a User
        // object representing Pat Smith is created and
        // appended to the Accounting group Users collection
        m_pUserNew->Groups->Append("Accounting");

        // Enumerate all User objects in the
        // catalog's Users collection.
        long lUsrIndex;
        long lGrpIndex;
        _variant_t vIndex;
        for (lUsrIndex=0; lUsrIndex<m_pCatalog->Users->Count; lUsrIn
        {
            vIndex = lUsrIndex;

```

```

m_pUser = m_pCatalog->Users->GetItem(vIndex);
cout<<"  "<<m_pUser->Name <<endl;
cout<<"  Belongs to these groups:"<<endl;

// Enumerate all Group objects in each User
// object's Groups collection.
if(m_pUser->Groups->Count != 0)
{
    for(lGrpIndex=0;lGrpIndex<m_pUser->Groups->Count;lGr
    {
        vIndex = lGrpIndex;
        m_pGroup = m_pUser->Groups->GetItem(vIndex);
        cout<<"      "<< m_pGroup->Name<<endl;
    }
}
else
    cout<<"      [None]"<<endl;
}

// Enumerate all Group objects in the default
// workspace's Groups collection.
for (lGrpIndex=0; lGrpIndex<m_pCatalog->Groups->Count; lGrpI
{
    vIndex = lGrpIndex;
    m_pGroup = m_pCatalog->Groups->GetItem(vIndex);
    cout<<"  "<< m_pGroup->Name <<endl;
    cout<<"  Has as its members:"<<endl;

    // Enumerate all User objects in each Group
    // object's Users Collection.
    if(m_pGroup->Users->Count != 0)
    {
        for (lUsrIndex=0; lUsrIndex<m_pGroup->Users->Count;
        {
            vIndex = lUsrIndex;
            m_pUser = m_pGroup->Users->GetItem(vIndex);
            cout<<"      "<<m_pUser->Name<<endl;
        }
    }
    else
        cout<<"      [None]"<<endl;
}

// Delete new User and Group object because this
// is only a demonstration.
m_pCatalog->Users->Delete("Pat Smith");
m_pCatalog->Groups->Delete("Accounting");
}
catch(_com_error &e)

```

```
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in include files...."<< endl;
}
}
// EndGroupCpp
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

Indexes Append Method Example (VC++)

The following code demonstrates how to create a new index. The index is on two columns in the table.

```
// BeginCreateIndexCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" no_name

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void CreateIndexX(void);

/////////////////////////////////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
/////////////////////////////////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    CreateIndexX();

    ::CoUninitialize();
}

/////////////////////////////////////////////////////////////////
//                                                                    //
//      CreateIndexX Function                                        //
//                                                                    //
/////////////////////////////////////////////////////////////////
void CreateIndexX(void)
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
```

```

_TablePtr m_pTable    = NULL;
_IndexPtr m_pIndex    = NULL;
_CatalogPtr m_pCatalog = NULL;

//Define other variables
_bstr_t strcn("Provider='Microsoft.JET.OLEDB.4.0';"
    "Data source = 'c:\\Program Files\\Microsoft Office\\"
    "Office\\Samples\\Northwind.mdb'");
try
{
    TESTHR(hr = m_pTable.CreateInstance(__uuidof(Table)));
    TESTHR(hr = m_pIndex.CreateInstance(__uuidof(Index)));
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog)));

    // Open the catalog.
    m_pCatalog->PutActiveConnection(strcn);

    // Define the table and append it to the catalog.
    m_pTable->Name = "MyTable";
    m_pTable->Columns->Append("Column1",adInteger,0);
    m_pTable->Columns->Append("Column2",adInteger,0);
    m_pTable->Columns->Append("Column3",adVarChar,50);
    m_pCatalog->Tables->Append(_variant_t((IDispatch *)m_pTable)
    printf("Table 'MyTable' is appended.\n");

    // Define a multi-column index.
    m_pIndex->Name = "multicolidx";
    m_pIndex->Columns->Append("Column1",adInteger,0);
    m_pIndex->Columns->Append("Column2",adInteger,0);

    // Append the index to the table.
    m_pTable->Indexes->Append(_variant_t((IDispatch *)m_pIndex))
    printf("Index 'multicolidx' is appended.\n");

    // Delete the table.
    m_pCatalog->Tables->Delete("MyTable");
    printf("Table 'MyTable' is deleted.\n");
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource,(LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error ocured in CreateIndexX...."<< endl;
}

```

```
    }  
}  
// EndCreateIndexCpp
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

IndexNulls Property Example (VC++)

This example demonstrates the [IndexNulls](#) property of an [Index](#). The code creates a new index and sets the value of **IndexNulls** based on user input. Then, the **Index** is appended to the **Employees Table** in the *Northwind Catalog*. The new **Index** is applied to a [Recordset](#) based on the **Employees** table, and the **Recordset** is opened. A new record is added to the **Employees** table, with a **Null** value in the indexed field. Whether this new record is displayed depends on the setting of the **IndexNulls** property.

```
// BeignIndexNullCpp
#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    rename("EOF", "EndOfFile")
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace

#include "iostream.h"
#include "stdio.h"
#include "conio.h"
#include "ADOXIndexNullsX.h"

// Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void IndexNullsX(_bstr_t);

////////////////////////////////////
//                                                                    //
//      Main Function                                                //
//                                                                    //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    printf("\nShow records having indexed field value = NULL? (Y/N):");
    char input = getche();

    if(toupper(input)=='Y')
    {
        IndexNullsX("Allow");
    }
}
```

```

    }
    else if(toupper(input)=='N')
    {
        IndexNullsX("Ignore");
    }
    else
    {
        exit(0);
    }

    ::CoUninitialize();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
//      IndexNullsX Function                                                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void IndexNullsX(_bstr_t strSel)
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _CatalogPtr m_pCatalog      = NULL;
    _IndexPtr m_pIndexNew       = NULL;

    // Define ADODB object pointers
    ADODB::_ConnectionPtr m_pCnn          = NULL;
    ADODB::_RecordsetPtr m_pRstEmployees = NULL;

    // Define other variables
    IADORecordBinding *picRs = NULL; // Interface Pointer Declared
    CEmployeeRs emprs;           // C++ Class Object

    // Define string variable.
    _bstr_t strCnn("Provider='Microsoft.JET.OLEDB.4.0';"
        "data source='c:\\Program Files\\Microsoft Office\\Office\\"
        "Samples\\Northwind.mdb'");

    try
    {
        TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection));
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));
        TESTHR(hr = m_pIndexNew.CreateInstance(__uuidof(Index)));
        TESTHR(hr = m_pRstEmployees.CreateInstance(
            __uuidof(ADODB::Recordset)));

        // Connect the catalog.

```

```

m_pCnn->Open(strCnn, "", "", NULL);
m_pCatalog->PutActiveConnection(_variant_t((IDispatch *)m_pC

// Append Country column to new index.
m_pIndexNew->Columns->Append("Country", adVarWChar, 0);
m_pIndexNew->Name = "NewIndex";

// Set IndexNulls based on user input
if(strcmp((LPSTR)strSel, "Allow")==0)
{
    m_pIndexNew->IndexNulls = adIndexNullsAllow;
}
else if(strcmp((LPSTR)strSel, "Ignore")==0)
{
    m_pIndexNew->IndexNulls = adIndexNullsIgnore;
}

// Append new index to Employees table
m_pCatalog->Tables->GetItem("Employees")->Indexes->Append(
    _variant_t((IDispatch *)m_pIndexNew));
m_pRstEmployees->Index = m_pIndexNew->Name;
m_pRstEmployees->Open("Employees",
    _variant_t((IDispatch *)m_pCnn),
    ADODB::adOpenKeyset, ADODB::adLockOptimistic,
    ADODB::adCmdTableDirect);

// Add a new record to the Employees table.
m_pRstEmployees->AddNew();
m_pRstEmployees->Fields->GetItem("FirstName")->Value =
    (_bstr_t) "Gary";
m_pRstEmployees->Fields->GetItem("LastName")->Value =
    (_bstr_t) "Haarsager";
m_pRstEmployees->Update();

// Bookmark the newly added record.
_variant_t varBookmark = m_pRstEmployees->Bookmark;

// Use the new index to set the order of the records.
m_pRstEmployees->MoveFirst();
printf("\n\nIndex = %s, ", (LPSTR) m_pRstEmployees->Index);
printf("IndexNulls = %d\n\n", m_pIndexNew->IndexNulls);
cout<<"Country          -          Name"<<endl;

// Open an IADORecordBinding interface pointer which
// we will use for binding Recordset to a class
TESTHR(hr = m_pRstEmployees->QueryInterface(
    __uuidof(IADORecordBinding), (LPVOID*)&picRs));

// Bind the Recordset to a C++ class here

```

```

TESTHR(hr = picRs->BindToRecordset(&emprs));

// Enumerate the Recordset.The value of the
// IndexNulls property will determine if the newly
// added record appears in the output.
while(!(m_pRstEmployees->EndOfFile))
{
    printf("%s      -      %s %s\n",
        emprs.lemp_CountryStatus == adFldOK ?
        emprs.m_szemp_Country : "[Null]",
        emprs.lemp_FirstNameStatus == adFldOK ?
        emprs.m_szemp_FirstName : "<NULL>",
        emprs.lemp_LastNameStatus == adFldOK ?
        emprs.m_szemp_LastName : "<NULL>");
    m_pRstEmployees->MoveNext();
}

// Delete new record because this is a demonstration.
m_pRstEmployees->Bookmark = varBookmark;
m_pRstEmployees->Delete(ADODB::adAffectCurrent);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in include files...."<< endl;
}

if (m_pRstEmployees)
{
    if (m_pRstEmployees->State == 1)
    {
        m_pRstEmployees->Close();
        m_pRstEmployees = NULL;
    }
}

// Delete new Index because this is a demonstration.
if (m_pCatalog)
    m_pCatalog->Tables->GetItem("Employees")->Indexes->
        Delete(m_pIndexNew->Name);

if (m_pCnn)

```

```

    {
        if (m_pCnn->State == 1)
        {
            m_pCnn->Close();
            m_pCnn = NULL;
        }
    }

    m_pCatalog = NULL;
}
// EndIndexNullCpp

```

IndexNullX.h

```

// BeginIndexNullsH
// IndexNullsX.h

#include "icrsint.h"

//This class extracts LastName, Country, FirstName from Employees tabl

class CEmployeeRs : public CADORecordBinding
{
BEGIN_ADO_BINDING(CEmployeeRs)
    // Column LastName is the 2nd field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szemp_LastName,
        sizeof(m_szemp_LastName), lemp_LastNameStatus, TRUE)
    // Column Country is the 11th field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(11, adVarChar, m_szemp_Country,
        sizeof(m_szemp_Country), lemp_CountryStatus, TRUE)
    // Column Country is the 17th field in the table
    ADO_VARIABLE_LENGTH_ENTRY2(17, adVarChar, m_szemp_FirstName,
        sizeof(m_szemp_FirstName), lemp_FirstNameStatus, TRUE)
END_ADO_BINDING()

public:
    CHAR m_szemp_LastName[21];
    ULONG lemp_LastNameStatus;
    CHAR m_szemp_Country[16];
    ULONG lemp_CountryStatus;
    CHAR m_szemp_FirstName[11];
    ULONG lemp_FirstNameStatus;
};
// EndIndexNullsH

```

ADOX 2.5 

Keys Append Method, Key Type, RelatedColumn, RelatedTable and UpdateRule Properties Example (VC++)

The following code demonstrates how to create a new foreign key. It assumes two tables (Customers and Orders) exist.

```
// BeginCreateKeyCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace
#import "c:\Program Files\Common Files\system\ado\msado15.dll"

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void CreateKeyX(void);

////////////////////////////////////
//                                     //
//      Main Function                   //
//                                     //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    CreateKeyX();

    ::CoUninitialize();
}

////////////////////////////////////
//                                     //
//      CreateKeyX Function             //
//                                     //
////////////////////////////////////
```

```

////////////////////////////////////
void CreateKeyX(void)
{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _KeyPtr m_pKeyForeign = NULL;
    _CatalogPtr m_pCatalog = NULL;

    //Define other variables
    _bstr_t strcnn("Provider='Microsoft.JET.OLEDB.4.0';"
        "Data source = 'c:\\Program Files\\Microsoft Office\\"
        "Office\\Samples\\Northwind.mdb';");
    try
    {
        TESTHR(hr = m_pKeyForeign.CreateInstance(__uuidof(Key));
        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog));
        m_pCatalog->PutActiveConnection(strcnn);

        // Define the foreign key.
        m_pKeyForeign->Name = "CustOrder";
        m_pKeyForeign->Type = adKeyForeign;
        m_pKeyForeign->RelatedTable = "Customers";
        m_pKeyForeign->Columns->Append("CustomerId",adVarChar,0);
        m_pKeyForeign->Columns->GetItem("CustomerId")->RelatedColumn
            "CustomerId";
        m_pKeyForeign->UpdateRule = adRICascade;

        // To pass as column parameter to Key's Append method
        _variant_t vOptional;
        vOptional.vt = VT_ERROR;
        vOptional.scode = DISP_E_PARAMNOTFOUND;

        // Append the foreign key.
        m_pCatalog->Tables->GetItem("Orders")->Keys->
            Append(_variant_t((IDispatch *)m_pKeyForeign),
                adKeyPrimary,vOptional,L"",L"");
        printf("Foreign key 'CustOrder' is added.\n");

        // Delete the key as this is a demonstration.
        m_pCatalog->Tables->GetItem("Orders")->Keys->
            Delete(m_pKeyForeign->Name);
        printf("Foreign key 'CustOrder' is deleted.\n");
    }

    catch(_com_error &e)
    {
        // Notify the user of errors if any.
    }
}

```

```
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}

catch(...)
{
    cout << "Error occured in include files...."<< endl;
}
}
// EndCreateKeyCpp
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 

NumericScale and Precision Properties Example (VC++)

This example demonstrates the [NumericScale](#) and [Precision](#) properties of the [Column](#) object. This code displays their value for the **Order Details** table of the *Northwind* database.

```
// BeginNumericScalePrecCpp
#import "c:\Program Files\Common Files\system\ado\msadox.dll" \
    no_namespace
#import "c:\Program Files\Common Files\system\ado\msado15.dll"

#include "iostream.h"
#include "stdio.h"
#include "conio.h"

//Function declarations
inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};
void NumericScalePrecX(void);

////////////////////////////////////
//                               //
//      Main Function              //
//                               //
////////////////////////////////////
void main()
{
    if(FAILED(::CoInitialize(NULL)))
        return;

    NumericScalePrecX();

    ::CoUninitialize();
}

////////////////////////////////////
//                               //
//      NumericScalePrecX Function //
//                               //
////////////////////////////////////
void NumericScalePrecX(void)
{
    HRESULT hr = S_OK;
```

```

// Define ADOX object pointers.
// Initialize pointers on define.
// These are in the ADOX:: namespace.
_CatalogPtr m_pCatalog = NULL;
_TablePtr m_pTable = NULL;
_ColumnPtr m_pColumn = NULL;

//Define ADODB object pointers
ADODB::_ConnectionPtr m_pCnn = NULL;

//Declare string variables
_bstr_t strCnn("Provider='Microsoft.JET.OLEDB.4.0';"
    "Data Source='c:\\Program Files\\Microsoft Office\\"
    "Office\\Samples\\Northwind.mdb'");
try
{
    TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection));
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog)));

    // Connect the catalog.
    m_pCnn->Open (strCnn, "", "", NULL);

    m_pCatalog->PutActiveConnection(variant_t((IDispatch *)m_pCnn));

    // Retrieve the Order Details table
    m_pTable = m_pCatalog->Tables->GetItem("Order Details");

    // Display numeric scale and precision of
    // small integer fields.
    _variant_t vIndex;
    for (long lIndex=0; lIndex < m_pTable->Columns->Count; lIndex++)
    {
        vIndex = lIndex ;
        m_pColumn = m_pTable->Columns->GetItem(vIndex);
        if(m_pColumn->Type == adSmallInt)
        {
            cout << "Column: " << m_pColumn->GetName() << endl;
            cout << "Numeric scale: " << (_bstr_t) m_pColumn
                GetNumericScale() << endl;
            cout << "Precision: " << m_pColumn->GetPrecision()
                << endl;
        }
    }
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
}

```

```
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error occured in NumericScalePrecX...."<< endl;
}
}
// EndNumericScalePrecCpp
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 


```

{
    HRESULT hr = S_OK;

    // Define ADOX object pointers.
    // Initialize pointers on define.
    // These are in the ADOX:: namespace.
    _CatalogPtr m_pCatalog = NULL;

    //Define ADODB object pointers.
    ADODB::_ConnectionPtr m_pCnn = NULL;
    ADODB::_CommandPtr m_pCommand = NULL;
    ADODB::_ParameterPtr m_pParameter = NULL;

    try
    {
        TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection));

        //Open the Connection
        m_pCnn->Open("Provider='Microsoft.JET.OLEDB.4.0';"
            "Data Source='c:\\Program Files\\Microsoft Office\\"
            "Office\\Samples\\Northwind.mdb';", "", "", NULL);

        TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));

        //Open the catalog
        m_pCatalog->PutActiveConnection(_variant_t((IDispatch *)m_pC));

        //Get the command object
        m_pCommand = m_pCatalog->Procedures->GetItem("CustomerById")
            GetCommand();

        _variant_t vIndex;

        //Retrieve Parameter information
        m_pCommand->Parameters->Refresh();
        for (long lIndex = 0; lIndex < m_pCommand->Parameters->Count
            lIndex++)
        {
            vIndex = lIndex;
            m_pParameter = m_pCommand->Parameters->GetItem(vIndex);
            cout << m_pParameter->Name << ":" << m_pParameter->Type
                "\\n" << endl;
        }
    }
    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());
    }
}

```

```
        printf("\n\tSource : %s \n\tdescription : %s \n ",
              (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
    }
    catch(...)
    {
        cout << "Error occured in ProcedureParametersX...."<< endl;
    }
}
// EndProcedureParametersCpp
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 


```

HRESULT hr = S_OK;

// Define ADOX object pointers.
// Initialize pointers on define.
// These are in the ADOX:: namespace.
_CatalogPtr m_pCatalog = NULL;
_TablePtr m_pTable = NULL;

// Define ADODB object pointers.
ADODB::_ConnectionPtr m_pCnn = NULL;

//Define string variables
_bstr_t strCnn("Provider='Microsoft.JET.OLEDB.4.0';"
    "Data Source='c:\\Program Files\\Microsoft Office\\"
    "Office\\Samples\\Northwind.mdb'");
try
{
    TESTHR(hr = m_pCnn.CreateInstance(__uuidof(ADODB::Connection));
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof (Catalog)));
    TESTHR(hr = m_pTable.CreateInstance(__uuidof (Table)));

    // Connect the catalog.
    m_pCnn->Open (strCnn, "", "", NULL);

    m_pCatalog->PutActiveConnection(variant_t((IDispatch *)m_pCnn);

    m_pTable->Name="MyContacts";
    m_pTable->ParentCatalog = m_pCatalog;

    // Create fields and append them to the new Table object.
    m_pTable->Columns->Append("ContactId", adInteger,0);

    // Make the ContactId column and auto incrementing column
    m_pTable->Columns->GetItem("ContactId")->Properties->
        GetItem("AutoIncrement")->Value = true;
    m_pTable->Columns->Append("CustomerID", adVarChar,0);
    m_pTable->Columns->Append("FirstName", adVarChar,0);
    m_pTable->Columns->Append("LastName", adVarChar,0);
    m_pTable->Columns->Append("Phone", adVarChar, 20);
    m_pTable->Columns->Append("Notes", adLongVarChar,0);
    m_pCatalog->Tables->Append(_variant_t((IDispatch*)m_pTable))

    // Refresh the database.
    m_pCatalog->Tables->Refresh();

    printf("Table 'MyContacts' is added.\n");

    // Delete new table, since this is only an example

```

```

        m_pCatalog->Tables->Delete("MyContacts");

        printf("Table 'MyContacts' is deleted.\n");
    }
    catch(_com_error &e)
    {
        // Notify the user of errors if any.
        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());

        printf("\n\tSource : %s \n\tdescription : %s \n ",
            (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
    }
    catch(...)
    {
        cout << "Error occurred in CreateAutoIncrColumnX...."<< endl;
    }

    m_pCatalog = NULL;
}
// EndCreateAutoIncrColumnCpp

```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADOX 2.5 


```

HRESULT hr = S_OK;

// Define ADOX object pointers.
// Initialize pointers on define.
// These are in the ADOX:: namespace.
_CatalogPtr m_pCatalog = NULL;
_TablePtr m_pTableNew = NULL;
_IndexPtr m_pIndexNew = NULL;
_IndexPtr m_pIndex = NULL;
_ColumnPtr m_pColumn = NULL;

//Define string variable
_bstr_t strcnn("Provider='Microsoft.JET.OLEDB.4.0';"
    "Data Source = 'c:\\Program Files\\"
    "Microsoft Office\\Office\\Samples\\Northwind.mdb'");

try
{
    TESTHR(hr = m_pCatalog.CreateInstance(__uuidof(Catalog)));
    TESTHR(hr = m_pTableNew.CreateInstance(__uuidof(Table)));
    TESTHR(hr = m_pIndexNew.CreateInstance(__uuidof(Index)));
    TESTHR(hr = m_pIndex.CreateInstance(__uuidof(Index)));
    TESTHR(hr = m_pColumn.CreateInstance(__uuidof(Column)));

    // Connect the catalog
    m_pCatalog->PutActiveConnection(strcnn);

    // Name new table
    m_pTableNew->Name = "NewTable";

    // Append a numeric and a text field to new table.
    m_pTableNew->Columns->Append("NumField", adInteger, 20);
    m_pTableNew->Columns->Append("TextField", adVarChar, 20);

    // Append new Primary Key index on NumField column
    // to new table
    m_pIndexNew->Name = "NumIndex";
    m_pIndexNew->Columns->Append("NumField", adInteger, 0);
    // here "-1" is required instead of "true".
    m_pIndexNew->PutPrimaryKey(-1);
    m_pIndexNew->PutUnique(-1);
    m_pTableNew->Indexes->Append(
        _variant_t ((IDispatch*)m_pIndexNew));

    // Append an index on Textfield to new table.
    // Note the different technique: Specifying index and
    // column name as parameters of the Append method
    m_pTableNew->Indexes->Append("TextIndex", "TextField");

    // Append the new table

```

```

m_pCatalog->Tables->Append(_variant_t ((IDispatch*)m_pTableN

cout << m_pTableNew->Indexes->Count << " Indexes in "
    << m_pTableNew->Name << " Table" << endl;
m_pCatalog->Tables->Refresh();

_variant_t vIndex;
// Enumerate Indexes collection.
for (long lIndex = 0;lIndex < m_pTableNew->Indexes->Count;
    lIndex++)
{
    vIndex = lIndex;
    m_pIndex = m_pTableNew->Indexes->GetItem(vIndex);
    cout << "Index " << m_pIndex->Name << endl;
    cout << "    Primary key = " << (m_pIndex->GetPrimaryKey(
        "True" : "False")) << endl;
    cout << "    Unique = " << (m_pIndex->GetUnique() ? "Tru
        "False") << endl;

    // Enumerate Columns collection of each Index
    // object.
    cout << "    Columns" << endl;

    for (long lIndex = 0;lIndex < m_pIndex->Columns->Count;
        lIndex++)
    {
        vIndex = lIndex ;
        m_pColumn = m_pIndex->Columns->GetItem(vIndex);
        cout << "        " << m_pColumn->Name << endl;
    }
}

// Delete new table as this is a demonstration
m_pCatalog->Tables->Delete(m_pTableNew->Name);
}
catch(_com_error &e)
{
    // Notify the user of errors if any.
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("\n\tSource : %s \n\tdescription : %s \n ",
        (LPCSTR)bstrSource, (LPCSTR)bstrDescription);
}
catch(...)
{
    cout << "Error ocured in PrimaryKeyX...."<< endl;
}

```

```
    m_pCatalog = NULL;  
}  
// EndPrimaryKeyCpp
```

[© 1998-2003 Microsoft Corporation. All rights reserved.](#)

ADO 2.5 

Legal and Copyright Information

Microsoft ActiveX Data Objects (ADO)

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

1998-2003 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows NT, ActiveX, Developer Studio, FoxPro, JScript, MSDN, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Apple is a registered trademark of Apple Computer, Inc.

Intel is a registered trademark of Intel Corporation.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

© 1998-2003 Microsoft Corporation. All rights reserved.