ADO Overview

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxADOOverviewC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxADOOverviewS"}

ActiveX Data Objects (ADO™) enables you to write a client application to access and manipulate data in a database server through a provider (database interface). ADO's primary benefits are ease of use, high speed, low memory overhead, and a small disk footprint. This help file is for ADODB, an implementation of ADO optimized for use with the OLE DB provider.

The ADO object model consists of a core set of interfaces for accessing all types of data. Because a single set of core interfaces may not serve all client/server developers, the ADO architecture can extend itself by dynamically creating objects to accommodate specific provider or client application needs. Such extensions derive their functionality by writing to ADO and to the provider directly. This mechanism is very flexible and permits considerable freedom of interaction between ADO and its extended objects; these extensions can supplement or override existing ADO objects, methods, and properties.

In ADO, the **Connection**, **Recordset**, and **Command** objects are the main interfaces to data. The minimal Microsoft® Visual Basic™ code to generate a **Recordset** is as follows:

```
Sub Main()

   ' Create a Recordset object.
   Dim rstMain as New ADODB.Recordset

   ' Open a Recordset object using the Source and
   ' ActiveConnection arguments.
   rstMain.Open "SELECT * FROM authors", _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"

End Sub
```

This generates a forward-only, read-only **Recordset** object. A slightly more functional **Recordset** can be generated as follows:

```
Sub Main()

   ' Create a Recordset object.
   Dim rstMain as New ADODB.Recordset

   ' Specify a keyset cursor.
   rstMain.CursorType = adOpenKeyset

   ' Set the locking for batch updating.
   rstMain.LockType = adConcurBatchOptimistic

   ' Open a Recordset object using the Source and
   ' ActiveConnection arguments.
   rstMain.Open "SELECT * FROM authors", _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"

End Sub
```

This creates a fully scrollable and batch-updatable Recordset.

Note This example is written in Microsoft Visual Basic. For applications that use Microsoft Visual Basic Script™ (for example, Microsoft ActiveX Server™), you need to convert the constants to their numeric values. These values can be found in the appropriate Help topics for the methods and properties shown.

In ADO, the object hierarchy is de-emphasized. Unlike DAO, you no longer have to navigate through a hierarchy to create objects because most ADO objects can be independently created. This allows you to create and track only the objects you need. This model also results in fewer ADO objects and thus a smaller working set.

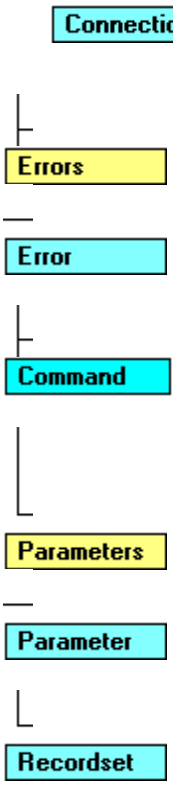
ADO supports key features for building client/server and web-based applications, including the following:

- Independently-created objects
- Batch updating
- Support for stored procedures with in/out parameters and return values
- Different cursor types including the potential for support of back-end–specific cursors
- Advanced recordset cache management
- Support for limits on number of returned rows and other query goals
- Support for multiple recordsets returned from stored procedures or batch statements
- Free-threaded objects for ISAPI applications

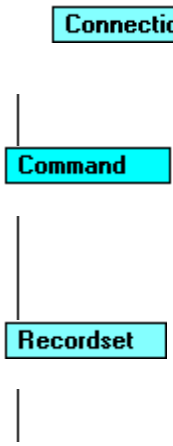
For a sample application using ADO, see the AdventureWorks web site. For more information on the OLE DB provider, see the OLE DB SDK web site.

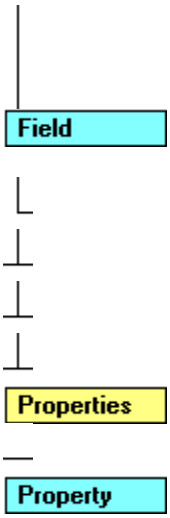
ADO Object Model

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxADOObjectModel10C"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxADOObjectModel10S"}



The **Connection**, **Command**, **Recordset**, and **Field** objects also have a **Properties** collection.




Field
Properties
Property

ADO Objects and Collections Reference

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxADOObjectsandCollectionsReferenceC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxADOObjectsandCollectionsReferenceS"}

In the following table, the type of collection in the first column contains the type of object in the second column. The third column describes what each type of object represents.

Collection	Object	Description
NA	**Command**	A prepared statement, such as an SQL statement.
NA	**Connection**	A connection to a data source.
Errors	**Error**	Information about errors resulting from provider operations.
Fields	**Field**	A column that is part of a recordset.
Parameters	**Parameter**	A parameter for a command.
Properties	**Property**	A provider-defined ("dynamic") property.
NA	**Recordset**	A set of records returned from a command or table.

ADO Methods by Object

{ewc HLP95EN.dll, DYNALINK, "See Also":"daidxADOMethodsByObjectC "} {ewc HLP95EN.dll, DYNALINK, "Specifics":"daidxADOMethodsByObjectS "}

This reference groups all ADO methods by object.

Command
Connection
Error
Field
Parameter
Property
Recordset

You can also find the ADO methods for the following collections.

Errors
Fields
Parameters
Properties

ADO Methods Reference

{ewc HLP95EN.dll, DYNALINK, "See Also":"daidxADOMethodsReferenceC "} {ewc HLP95EN.dll, DYNALINK, "Specifics":"daidxADOMethodsReferenceS "}

This reference alphabetically lists all ADO methods.

AddNew
Append
AppendChunk
BeginTrans
CancelUpdate
CancelUpdateBatch
Clear
Clone
Close
CommitTrans
CreateParameter
Delete
Execute
GetChunk
GetRows
Move
MoveFirst
MoveLast
MoveNext
MovePrevious
NextRecordset
Open
Refresh
Requery
Resync
RollbackTrans
Supports
Update
UpdateBatch

ADO Properties by Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxADOPropertiesByObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxADOPropertiesByObjectS"}

This reference groups all ADO properties by object.

Command
Connection
Error
Field
Parameter
Property
Recordset

You can also find the ADO properties for the following collections.

Errors
Fields
Parameters
Properties

ADO Properties Reference

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxADOPropertiesReferenceC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxADOPropertiesReferenceS"}

This reference alphabetically lists all ADO properties.

A-C

AbsolutePage
AbsolutePosition
ActiveConnection
ActualSize
Attributes
BOF
Bookmark
CacheSize
CommandText
CommandTimeout
CommandType
ConnectionString
ConnectionTimeout
Count
CursorType

D-N

DefaultDatabase
DefinedSize
Description
Direction
EditMode
EOF
Filter
HelpContext
HelpFile
IsolationLevel
LockType
MaxRecords
Mode
Name
NativeError
Number

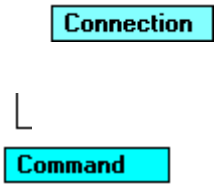
O-Z

OriginalValue
PageCount
PageSize
Prepared
Precision
Provider
RecordCount
Scale
Size
Source
SQLState
Status
Type
UnderlyingValue
Value
Version

Command Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjCommandC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjCommandX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjCommandP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjCommandM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjCommandS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":":1}

A **Command** object is a definition of a specific command that you intend to execute against a data source.



Remarks

You can create **Command** objects independently of a previously defined **Connection** object.

Use **Command** objects to return records and create a **Recordset** object, to execute a bulk operation, or to manipulate the structure of a database. Depending on the functionality the provider exposes, some collections, methods, or properties of a **Command** object may not be available.

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

- Associate an open connection and the **Command** object with the **ActiveConnection** property to execute the command.
- Define the text version of the command (for example, an SQL statement) with the **CommandText** property.
- Set the number of seconds a provider will wait for a command to execute with the **CommandTimeout** property.
- Specify the type of command described in the **CommandText** property with the **CommandType** property prior to execution in order to optimize performance.
- Determine whether or not the provider will save a prepared version of a command with the **Prepared** property.
- Manage arguments passed to and from the provider with the **Parameters** collection.
- Execute a command and return a **Recordset** object if appropriate with the **Execute** method.

For a complete list of all collections, methods, and properties available on a **Command** object, see the **Summary** topic.

Command Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumCommandC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumCommandS"}

A **Command** object contains these collections, these methods, and these properties.

Collections

Parameters (default)

Properties

Methods

CreateParameter

Execute

Properties

ActiveConnection

CommandText

CommandTimeout

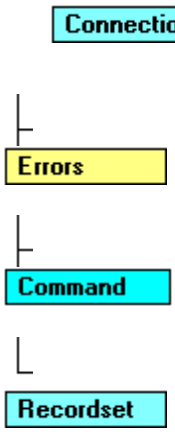
CommandType

Prepared

Connection Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjConnectionP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjConnectionM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjConnectionS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjConnectionU":1}

A **Connection** object is an object that represents an open connection to an OLE DB data source.



Remarks

You can create **Connection** objects independently of any other previously defined object.

A **Connection** object is a non-persistent object that represents an open session with a data source. In the case of a client/server database system, it literally maps to an underlying network connection to the server. Depending on the functionality the provider exposes, some collections, methods, or properties of a **Connection** object may not be available.

Using the collections, methods, and properties of a **Connection** object, you can do the following:

- Configure the connection before opening it with the **ConnectionString** and **ConnectionTimeout** properties.
- Choose among various cursor libraries accessible to the provider with the **CursorLocation** property.
- 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.
- Set the permissions for the connection with the **Mode** property.
- Select 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 against the connection with the **Execute** method and **CommandTimeout** property.
- 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 in use with the **Version** property.

For a complete list of all collections, methods, and properties available on a **Connection** object, see the **Summary** topic.

Connection Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumConnectionS"}

A **Connection** object contains these collections, methods, and properties.

Collections

Errors

Properties (default)

Methods

BeginTrans

Close

CommitTrans

Execute

Open

RollbackTrans

Properties

Attributes

CommandTimeout

ConnectionString (default)

ConnectionTimeout

DefaultDatabase

IsolationLevel

Mode

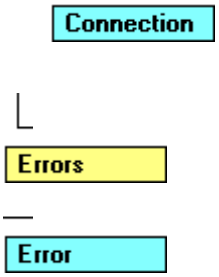
Provider

Version

Error Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjErrorM"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjErrorU":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjErrorP"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjErrorS"}

An **Error** object contains details about data access errors pertaining to a single operation involving ADO.



Remarks

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 alert that will be displayed on the screen if the error is not trapped.
- 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 **HelpFile** and **HelpContext** properties, which indicate the appropriate Microsoft Windows Help file and Help topic, respectively, (if any exist) for the error.

When a provider error occurs, it is placed in the **Errors** collection of the **Connection** object. If there is no valid **Connection** object, you will need to retrieve error information from the Visual Basic for Applications **Err** object. ADO supports the return of multiple errors by a single ADO operation to allow for error information specific to the provider. For example, this is how an error would be processed when you use ADO with OLE DB:

1. The client makes an ADO call.
2. ADO makes a call to a provider (via OLE DB).
3. The provider encounters an error.
1. Using the OLE DB SDK **Error** object, the provider creates an **OLE Error Info** object and puts an error record into it.
1. The client receives the error(s). By using the **OLE Error Info** object directly (through Visual Basic for Applications) or the ADO **Errors** collection, the client can retrieve detailed error information.

ADO can return the following specific errors:

Constant Name	Number	Description
adErrInvalidArgument	3001	Invalid argument.
adErrIllegalOperation	3219	Invalid operation.
adErrInTransaction	3246	Transaction error.
adErrFeatureNotAvailable	3251	Operation is not supported for this type of object.
adErrItemNotFound	3265	Item not found in this collection.
adErrObjectNotSet	3420	Object is invalid or not set.

adErrDataConversion	3421	Data type conversion error.

Just as providers do, ADO clears the **OLE Error Info** object before making a call that could potentially generate a new error. However, the **Errors** collection on the **Connection** object is cleared and populated only when ADO or the provider generates a new error. ADO only puts errors from the provider in the **Errors** collection; it does not add ADO-specific errors to the collection.

Some properties and methods return warnings which appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the **Delete**, **Resync**, **UpdateBatch** or **CancelUpdateBatch** methods on a **Recordset** object or before you set the **Filter** property on a **Recordset** object, call the **Clear** method on the **Errors** collection so that you can read the **Count** property of the **Errors** collection to test for returned warnings.

For a complete list of all collections, methods, and properties available on an **Error** object, see the **Summary** topic.

Errors Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolErrorM"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolErrorU":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolErrorP"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolErrorS"}

The **Errors** collection contains all stored **Error** objects, all of which pertain to a single operation involving ADO.

Remarks

Any operation involving ADO objects can generate one or more errors. As each error occurs, one or more **Error** objects may 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 may be placed in the **Errors** collection. 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. ADO only puts errors from the provider in the **Errors** collection; it does not add ADO-specific errors to the collection.

The set of **Error** objects in the **Errors** collection describes one error. 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 which appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the **Delete**, **Resync**, **UpdateBatch** or **CancelUpdateBatch** methods on a **Recordset** object or before you set the **Filter** property on a **Recordset** object, call the **Clear** method on the **Errors** collection so that 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.

For a complete list of all collections, methods, and properties available on an **Errors** collection, see the **Summary** topic.

Error Object, Errors Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumErrorS"}

Error Object

An **Error** object contains these properties:

Properties
Description (default)
HelpContext
HelpFile
NativeError
Number
Source
SQLState

Errors Collection

The **Errors** collection appears in each **Connection** object, and contains this method and this property:

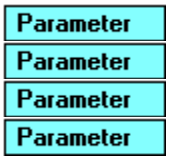
Method
Clear
Refresh

Property
Count

Field Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjFieldM"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjFieldU":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjFieldP"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjFieldS"}

A **Field** object represents a column of data with a common data type and a common set of properties.



Remarks

A **Recordset** object has a **Fields** collection made up of **Field** objects. 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 a **Recordset** with the **Value** field.
- Return the basic characteristics of a field with the **Type**, **Precision**, and **Scale** 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.
- Manipulate fields containing long binary data with the **AppendChunk** and **GetChunk** methods.
- Resolve discrepancies in field values during batch updating with the **OriginalValue** and **UnderlyingValue** properties.

To refer to a **Field** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

recordset.**Fields**(*0*)
recordset.**Fields**(*"name"*)
recordset.**Fields**!*name*
recordset(*"name"*)

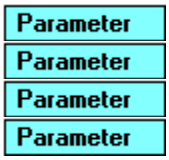
You can only use the exclamation point notation (!) to set or return the value of a **Field** object.

For a complete list of all collections, methods, and properties available on a **Field** object, see the **Summary** topic.

Fields Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolFieldC"}
{ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolFieldP"}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolFieldS"}
HLP95EN.DLL,DYNALINK,"Summary":"dacolFieldU":1}
{ewc HLP95EN.DLL,DYNALINK,"Example":"dacolFieldX":1}
{ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolFieldM"}
{ewc

A **Fields** collection contains all stored **Field** objects of a **Recordset** object.



Remarks

A **Recordset** object has a **Fields** collection made up of **Field** objects. Each **Field** object corresponds to a column in the **Recordset**.

To refer to a **Field** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

recordset.**Fields**(*0*)
recordset.**Fields**(*"name"*)
recordset.**Fields**!*name*
recordset(*"name"*)

You can only use the exclamation point notation (!) to set or return the value of a **Field** object.

Note See the **Field** object topic for a more detailed explanation of how to use **Field** objects.

For a complete list of all collections, methods, and properties available on a **Fields** collection, see the **Summary** topic.

Field Object, Fields Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumFieldS"}

Field Object

A **Field** object contains this collection, these methods and properties.

Collection

Properties

Methods

AppendChunk
GetChunk

Properties

ActualSize
Attributes
DefinedSize
Name
OriginalValue
Precision
Scale
Type
UnderlyingValue
Value

Fields Collection

A **Fields** collection appears in each **Recordset** object, and contains this method and this property.

Method

Refresh

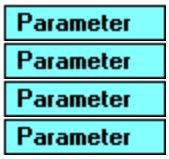
Property

Count

Parameter Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjParameterX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjParameterP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjParameterM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjParameterS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjParameterU":1}

A **Parameter** object represents a parameter or argument associated with a **Command** object based on a parameterized query or stored procedure.



Remarks

Parameter objects represent parameters associated with parameterized queries, or the in/out arguments or return values of stored procedures. Depending on the functionality the provider exposes, 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.
- Set or return parameter characteristics with the **Attributes** and **Direction**, **Precision**, **Scale**, **Size**, and **Type** properties.
- Pass long binary data to a parameter with the **AppendChunk** method.

If you know the names and properties of the parameters associated with the stored procedure or parameterized query you wish to call, you can 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.

To refer to a **Parameter** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

command.**Parameters**(*0*)
command.**Parameters**(*"name"*)
command.**Parameters**!*name*
command(*"name"*)

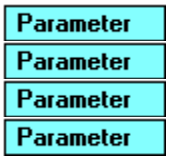
You can only use the exclamation point notation (!) to set or return the value of a **Parameter** object.

For a complete list of all collections, methods, and properties available on a **Parameter** object, see the **Summary** topic.

Parameters Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolParameterX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolParameterP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolParameterM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolParameterS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolParameterU":1}

A **Parameters** collection contains all the **Parameter** objects of a **Command** object.



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-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.

If 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 names and properties of the parameters associated with the stored procedure or parameterized query you wish to call. 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 use parameters at all. Use the **Delete** method to remove **Parameter** objects from the **Parameters** collection if necessary.

To refer to a **Parameter** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

command.**Parameters**(*0*)
command.**Parameters**(*"name"*)
command.**Parameters**!*name*
command(*"name"*)

You can only use the exclamation point notation (!) to set or return the value of a **Parameter** object.

For a complete list of all collections, methods, and properties available on a **Parameters** collection, see the **Summary** topic.

Parameter Object, Parameters Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumParameterS"}

Parameter Object

A **Parameter** object contains this collection, this method and these properties.

Collection

Properties

Method

AppendChunk

Properties

Attributes
Direction
Name
Precision
Scale
Size
Type
Value

Parameters Collection

A **Parameters** collection appears in each **Command** object and contains these methods and this property.

Methods

Append
Delete
Refresh

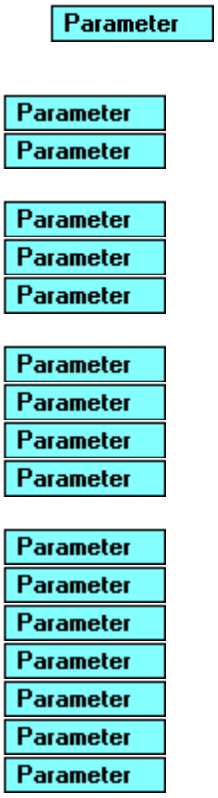
Property

Count

Property Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjPropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjPropertyP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjPropertyM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjPropertyS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjPropertyU":1}

A **Property** object represents a dynamic characteristic of an ADO object that is defined by the provider.



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. However, these built-in properties do not appear as **Property** objects in an object's **Properties** collection, so while you can change their values, you cannot modify their characteristics or delete them.

Many OLE DB providers will expose additional object properties to ADO. These dynamic properties provide information about additional functionality available from the provider. 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.

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

- The **Name** property, a string that uniquely identifies the property.
- The **Type** property, an integer that specifies the property data type.
- The **Value** property, a variant that contains the property setting.
- The **Attributes** property, a long value that indicates characteristics of the property specific to the provider.

To refer to a **Property** object in a collection by its **Name** property setting, use the following syntax

form:

object.**Properties**("*name*")

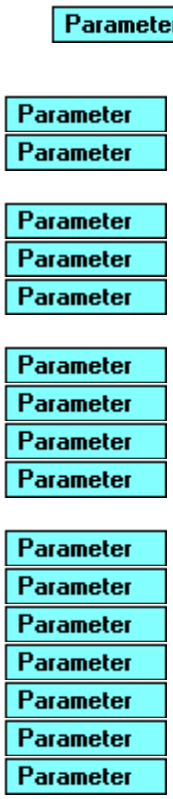
With the same syntax form, you can also refer to the **Value** property of a **Property** object. The context of the reference will determine whether you are referring to the **Property** object itself or the **Value** property of the **Property** object.

For a complete list of all collections, methods, and properties available on a **Property** object, see the **Summary** topic.

Properties Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolPropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolPropertyP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolPropertyM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolPropertyS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolPropertyU":1}

A **Properties** collection contains all the **Property** objects for a specific instance of an object.



Remarks

All 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.

To refer to a **Property** object in a collection by its **Name** property setting, use the following syntax form:

object.**Properties**(*"name"*)

Note See the **Property** object topic for a more detailed explanation of how to use **Property** objects.

For a complete list of all collections, methods, and properties available on a **Property** collection, see the **Summary** topic.

Property Object, Properties Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumPropertyS"}

Property Object

A **Property** object contains these properties.

Properties

Attributes

Name

Type

Value

Properties Collection

A **Properties** collection appears in the **Connection**, **Command**, **Parameter**, **Recordset**, and **Field** objects, and contains this method and this property.

Method

Refresh

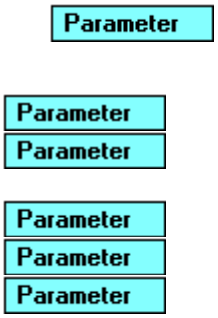
Property

Count

Recordset Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjRecordsetU":1}

A **Recordset** object is a data cursor that represents records from a base table or the results from a command.



Remarks

You can create **Recordset** objects independently of a previously defined **Connection** object.

You use **Recordset** objects to manipulate data from a provider at the record level. When you use ADO, you manipulate data almost entirely using **Recordset** objects. All **Recordset** objects are constructed using records (rows) and fields (columns). Depending on the functionality the provider exposes, some collections, methods, or properties of a **Recordset** object may not be available.

You can use one of four different cursor types when opening a **Recordset** object:

- **Dynamic cursor** — allows you to view additions, changes, and deletions by other users, and allows all types of movement through the **Recordset**; allows bookmarks if 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 from your recordset. Data changes by other users will still be visible.
- **Static cursor** — provides a static copy of a set of records for you to use to find data or generate reports. Additions, changes, or deletions by other users will not be visible.
- **Forward-only cursor** — behaves identically to a static cursor except that it only allows you to scroll forward through records. This improves performance in situations where you only need to make a single pass through a recordset.

Set the **CursorType** property prior to opening the **Recordset** object to choose the cursor type of the **Recordset** object.

If you don't specify a cursor type, ADO attempts to use the cursor with the fastest query response, starting with forward-only, and moving if necessary to static, keyset, and dynamic cursors.

You can create as many **Recordset** objects as needed. Different **Recordset** objects can access the same tables and fields without conflicting.

When you create a **Recordset** object, the current record is positioned to the first record (if any) and the **BOF** property is set to **True**. If there are no records, the **RecordCount** property setting is 0, and the **BOF** and **EOF** property settings are **True**.

You can use the **MoveNext**, **MovePrevious**, **MoveFirst**, and **MoveLast** methods, the **Move** method, and the **AbsolutePosition** and **AbsolutePage** 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 check for the beginning or end of the **Recordset** object.

Recordset objects may 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 cache changes to more than one record and then transmit them in a single call to the provider with the **UpdateBatch** method. This applies to changes made with the **AddNew**, **Update**, **CancelUpdate**, 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.

For a complete list of all collections, methods, and properties available on a **Recordset** object, see the **Summary** topic.

Recordset Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumRecordsetS"}

A **Recordset** object contains these collections, methods, and properties.

Collections

Fields (default)

Properties

Methods

Note The methods available on a particular **Recordset** depend on the provider and the parameters used to create the **Recordset.**

AddNew
CancelUpdate
CancelUpdateBatch
Clone
Close
Delete
GetRows
Move
MoveFirst
MoveLast
MoveNext
MovePrevious
NextRecordset
Open
Requery
Resync
Supports
Update
UpdateBatch

Properties

Note The properties available on a particular **Recordset** depend on the provider and the parameters used to create the **Recordset**.

AbsolutePage
AbsolutePosition
ActiveConnection
BOF
Bookmark
CacheSize
CursorType
EditMode
EOF
Filter

LockType
MaxRecords
PageCount
PageSize
RecordCount
Source
Status

AddNew Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthAddNewC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthAddNewX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthAddNewA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthAddNewS"}

Creates a new record for an updatable **Recordset** object.

Syntax

recordset.**AddNew** *Fields*, *Values*

The **AddNew** method syntax has these parts:

Part	Description
recordset	An object variable representing the **Recordset** object to which you want to add a new record.
Fields	Optional. A **Variant** representing a single name or a **Variant** array representing names of the field(s) in the new record.
Values	Optional. A **Variant** representing a single value or a **Variant** array representing values for the field(s) in the new record.

Remarks

Use the **AddNew** method to create and initialize a new record. Use the **Supports** method to verify whether you may add records to the current **Recordset** object.

After you call the **AddNew** method, the new record is added to the end of the **Recordset** and becomes the current record.

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

If *Fields* 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 behavior of the **AddNew** method depends on the updating mode of the **Recordset** object and whether or not you pass the *Fields* and *Values* arguments.

In immediate update mode, calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. ADO 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 *Fields* and *Values* arguments, ADO immediately posts the new record to the database (no **Update** call is necessary) and resets the **EditMode** property to **adEditNone**.

In batch update mode, calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. ADO 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 *Fields* and *Values* arguments, ADO sends the new record to the provider for storage in a cache and resets the **EditMode** property to **adEditNone**; you need to call the **UpdateBatch** method to post the new record to the underlying database.

Append Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthAppendC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthAppendX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthAppendA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthAppendS"}

Appends an object to a collection.

Syntax

Collection.**Append** *Object*

The *Collection* placeholder represents the collection to which you want to append an object. The *Object* placeholder is an object variable representing the object you wish to append.

Remarks

Use the **Append** method on a collection to add an object to that collection. This method is available only on the **Parameters** collection of a **Command** object. You must set the **Name** and **Type** properties of a **Parameter** object before appending it to the **Parameters** collection. If you select a numeric data type, you must also set the **Precision** property to a value greater than zero.

You can minimize calls to the provider to improve performance when using stored procedures or parameterized queries. However, you must know the names and properties of the parameters associated with the stored procedure or parameterized query you wish to call. 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 use parameters at all.

AppendChunk Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthAppendChunkC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthAppendChunkX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthAppendChunkA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthAppendChunkS"}

Appends data to a large text or binary data **Field** or **Parameter** object.

Syntax

{*field* | *parameter*}**.AppendChunk** *Data*

The **AppendChunk** method syntax has these parts.

Part	Description
field	An object variable representing a **Field** object in the **Fields** collection of an open **Recordset** object.
parameter	An object variable representing a **Parameter** object in the **Parameters** collection of a **Command** object.
Data	A **Variant** containing the data you want to append to *field* or *parameter.*

Remarks

Use the **AppendChunk** method on a **Field** or **Parameter** object to fill it with long binary data. If system memory is limited, the **AppendChunk** method allows you to manipulate long values in portions rather than in their entirety.

Field

The presence of the **adFldLong** constant in the **Attributes** property of a **Field** object indicates that 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 there is no current record when you call **AppendChunk** on a **Field** object, an error occurs.

Parameter

Each **AppendChunk** call on a **Parameter** object appends to existing parameter data. An **AppendChunk** call that passes a **Null** value empties the parameter value.

BeginTrans, CommitTrans, RollbackTrans Methods

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthBeginTransC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthBeginTransX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthBeginTransA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthBeginTransS"}

The 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

[**Set** *Level* =] *connection*.{**BeginTrans**|**CommitTrans**|**RollbackTrans**}

The *connection* placeholder is an object variable representing an open **Connection** object.

The *Level* placeholder represents a **Long** variable to which the **BeginTrans** method can return a value indicating the nesting level of the transaction. The **CommitTrans** and **RollbackTrans** methods do not return any value.

Remarks

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 goes through. Also, try grouping operations that require disk access into transaction blocks to improve the performance of your application. This buffers your operations and may reduce the number of times the disk is accessed.

Once you call the **BeginTrans** method, the provider will no longer instantaneously commit any 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. You must resolve the most recently opened transaction before resolving 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 new a transaction. If the **Attributes** property is set to **adXactCommitRetaining**, ADO automatically starts a new transaction after a **CommitTrans** call. If the **Attributes** property is set to **adXactAbortRetaining**, ADO automatically starts a new transaction after a **RollbackTrans** call.

CancelUpdate Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCancelUpdateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCancelUpdateX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCancelUpdateA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCancelUpdateS"}

Cancels any changes made to the current record or to a new record.

Syntax

recordset.**CancelUpdate**

The *recordset* placeholder is an object variable representing an open **Recordset** object.

Remarks

Use the **CancelUpdate** method to cancel any changes made to the current record or to discard a newly added record.

If you are adding a new record when you call the **CancelUpdate** method, the record that was current prior to the **AddNew** call becomes the current record again.

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

CancelUpdateBatch Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCancelUpdateBatchC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCancelUpdateBatchX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCancelUpdateBatchA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCancelUpdateBatchS"}

Cancels a pending batch update.

Syntax

recordset.**CancelUpdateBatch** *Affect*

The **CancelUpdateBatch** method syntax has these parts:

Part	Description
recordset	An object variable representing an open **Recordset** object.
Affect	Optional. A **Long** expression that determines on how many records the **CancelUpdateBatch** method should act. Can be one of the following constants: • **adAffectCurrent**, 1 — Cancel changes only for the current record. • **adAffectAll**, 2 (Default) — Cancel changes for all the records in the **Recordset** object. • **adAffectGroupFilter**, ??? — Cancel changes for the records that satisfy the current **Filter** property setting. There must be a valid **Filter** on the **Recordset** in order to use this constant.

Remarks

Use the **CancelUpdateBatch** method to cancel any pending updates in a recordset in batch update mode. If the recordset is in immediate update mode, calling **CancelUpdateBatch** generates an error.

If you are editing the current record or are adding a new record when you call **CancelUpdateBatch**, 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 **CancelUpdateBatch** 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 **CancelUpdateBatch** 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.

Clear Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthClearC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthClearX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthClearA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthClearS"}

Removes all of the objects in a collection.

Syntax

Collection.**Clear**

The *Collection* placeholder represents the collection whose objects you want to remove.

Remarks

Use the **Clear** method on the **Errors** collection to remove all existing **Error** objects from the collection. Some properties and methods return warnings which appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the **Delete**, **Resync**, **UpdateBatch** or **CancelUpdateBatch** methods on a **Recordset** object or before you set the **Filter** property on a **Recordset** object, call the **Clear** method on the **Errors** collection so that you can read the **Count** property of the **Errors** collection to test for returned warnings.

Clone Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCloneC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCloneX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCloneS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCloneA"}

Creates a duplicate **Recordset** object from an existing **Recordset** object.

Syntax

Set *duplicate* = *original*.**Clone**

The **Clone** method syntax has these parts:

Part	Description
duplicate	An object variable identifying the duplicate **Recordset** object you're creating.
original	An object variable identifying the **Recordset** object you want to duplicate.

Remarks

Use the **Clone** method to create multiple, duplicate **Recordset** objects, particularly if you want to be able 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 undefined. Changes you make to one **Recordset** object are visible in all of its clones regardless of cursor type. Closing either the original recordset or any of its cloned copies does not close any of the other copies.

You can only clone a **Recordset** object that supports bookmarks. You can use bookmark references from one **Recordset** object in any of its clones to access the same records.

Close Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCloseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCloseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCloseS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCloseA"}

Closes an open object and any dependent objects.

Syntax

object.**Close**

The *object* placeholder is an object variable representing an open **Connection** or **Recordset** object.

Remarks

Use the **Close** method to close either a **Connection** object or a **Recordset** object to free any associated system resources.

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 **Null**. Also, its **Parameters** collection will be cleared.

You can later call the **Open** method to reestablish 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. Closing a **Connection** object while a transaction is in progress generates an error.

Recordset

Using the **Close** method to close a **Recordset** object releases any locks on the associated data. You can later call the **Open** method to reopen the recordset with the same or modified attributes. While the **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. If you close the **Recordset** object during batch updating, all unposted changes 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.

CreateParameter Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateParameterX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateParameterA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateParameterS"}

Creates a new **Parameter** object with the specified properties.

Syntax

Set *parameter* = *command*.**CreateParameter**(*Name*, *Type*, *Direction*, *Value*)

The **CreateParameter** method syntax has these parts:

Part	Description
parameter	An object variable representing the **Parameter** object you want to create.
command	An object variable representing the **Command** object for whose **Parameters** collection you want to create a new **Parameter** object.
Name	A **String** representing the name of the **Parameter** object.
Type	Optional. A **Long** value specifying the data type of the **Parameter** object. See the **Type** property for valid settings.
Direction	Optional. A **Long** value specifying the type of **Parameter** object. See the **Direction** property for valid settings.
Value	Optional. A **Variant** specifying the value for the **Parameter** object.

Remarks

Use the **CreateParameter** method to create a new **Parameter** object with the specified name, type, direction 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.

Delete Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthdeleteC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthDeleteX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthDeleteA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthDeleteS"}

Deletes the current record in an open **Recordset** object or an object from a collection.

Syntax

For a **Recordset** object:

recordset.**Delete** *Affect*

The **Delete** method syntax for **Recordset** objects has these parts:

Part	Description
recordset	An object variable representing an open **Recordset** object.
Affect	Optional. A **Long** expression that determines on how many records the **Delete** method should act. Can be one of the following constants: • **adAffectCurrent**, 1 (Default) — Delete only the current record. • **adAffectAll**, 2 — Delete all the records in the **Recordset** object. • **adAffectGroupFilter**, ??? — Delete the records that satisfy the current **Filter** property setting. There must be a valid **Filter** on the **Recordset** in order to use this constant.

For a collection:

Collection.**Delete** *Object*

The *Collection* placeholder represents the collection from which you want to delete an object. The *Object* placeholder is a **String** representing the name of the object you wish to delete.

Remarks

Use the **Delete** method with a **Recordset** object to remove the current record or with a collection to remove one of its objects.

Recordset

Using the **Delete** method marks the current record in a **Recordset** object for removal. 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. Otherwise, the records are marked for deletion from the cache and the actual deletion happens when you call the **UpdateBatch** method.

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.

Deleted records can be recovered if you nest the deletes in a transaction and use the **RollbackTrans** method or if you are in batch update mode and use the **CancelUpdateBatch** 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.

Collection

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 when calling the **Delete** method — an object variable is not a valid argument.

Execute Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthExecuteC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthExecuteX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthExecuteA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthExecuteS"}

- **Command** — Executes the query, SQL statement, or stored procedure specified in the **CommandText** property.
- **Connection** — Executes the specified query, SQL statement, or stored procedure.

Syntax

On a **Command** object:

Set *recordset* **=** *command***.Execute**(*RecordsAffected***,** *Parameters*)
*command***.Execute** *RecordsAffected***,** *Parameters*

On a **Connection** object:

Set *recordset* **=** *connection***.Execute**(*CommandText***,** *RecordsAffected*)
*connection***.Execute** *CommandText***,** *RecordsAffected*

The **Execute** method syntax has these parts.

Part	Description
recordset	An object variable representing the **Recordset** object in which the results of the query are stored.
command	An object variable representing a **Command** object whose **CommandText** property contains the query to execute.
connection	An object variable representing a **Connection** object on which the query is executed.
RecordsAffected	Optional. A **Long** variable to which the provider returns the number of records that the operation affected.
Parameters	Optional. A **Variant** array of parameter values passed with an SQL statement.
CommandText	A **String** containing the SQL statement, query, or stored procedure to execute.

Remarks

Use the **Execute** method to execute an existing **Command** object or a query of your choosing. You can also specify a **Recordset** object in which to store the results if any.

Command

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 the execution generates are stored in a new **Recordset** object. If the command is not a row-returning query, the provider does not create a **Recordset** object and only returns a **Null** object reference. Most 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 `varArray(var1,,,var4)` as the *parameters* argument.

Connection

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 the execution generates are stored in a new **Recordset** object. If the command is not a row-returning query, the provider does not create a **Recordset** object and only returns a **Null** object reference. Most application languages allow you to ignore this return value if no **Recordset** is desired.

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.

GetChunk Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthGetChunkC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthGetChunkX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthGetChunkA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthGetChunkS"}

Returns all or a portion of the contents of a large **Field** object.

Syntax

variable = *field*.**GetChunk**(*Bytes*)

The **GetChunk** method syntax has these parts.

Part	Description
variable	A **String** or **Variant** variable that receives the data.
field	An object variable representing a **Field** object in the **Fields** collection of an open **Recordset** object.
Bytes	A **Long** expression equal to the number of bytes you want to retrieve.

Remarks

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

The bytes a **GetChunk** call returns are assigned to *variable*. If *Bytes* is greater than the number of bytes of remaining data, the **GetChunk** method returns only the remaining bytes without padding the data with empty spaces. If the field is empty, the **GetChunk** method returns **Null**.

The presence of the **adFldLong** constant in the **Attributes** property of a **Field** object indicates that 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, an error occurs.

GetRows Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthGetRowsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthGetRowsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthGetRowsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthGetRowsS"}

Retrieves multiple records of a **Recordset** into an array.

Syntax

Array = *recordset*.**GetRows**(*Rows, Bookmark, Fields*)

The **GetRows** method syntax has the following parts.

Part	Description
Array	A **Variant** variable in which to store the returned data.
recordset	An object variable representing a **Recordset** object.
Rows	A **Long** expression indicating the number of records to retrieve. Default is **adGetRowsRest**, -1.
Bookmark	Optional. A **String** or **Variant** that evaluates to the bookmark for the record from which the **GetRows** operation should begin.
Fields	Optional. A **Variant** 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. The returned data is read-only.

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.

If you want to restrict the fields 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.

Move Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthMoveC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthMoveX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthMoveA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthMoveS"}

Moves the position of the current record in a **Recordset** object.

Syntax

recordset.**Move** *Rows*, *StartBookmark*

The **Move** method syntax has these parts.

Part	Description
recordset	An object variable representing the **Recordset** object whose current record position you want to move.
Rows	A signed **Long** expression specifying the number of records the current record position moves.
StartBookmark	Optional. A **String** or **Variant** that evaluates to a bookmark.

Remarks

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

If the *Rows* argument is greater than zero, the current record position moves forward (toward the end of the recordset). If *Rows* is less than zero, the current record position moves backward (toward the beginning of the recordset). If *Rows* is zero, the current record is refreshed from the provider or the local cache.

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 *StartBookmark* argument, the move is relative to the record with this bookmark. 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 *Rows* that moves the current record position outside of 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. ADO will also retrieve a new set of records if you are using a local cache and you pass the *StartBookmark* argument.

If the **Recordset** object is forward-only, a user can still pass a *Rows* less than zero as long as 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 create a "cached cursor" that supports full scrolling over a provider that only supports forward scrolling. Because a cached cursor will load all records 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 still generates an error.

MoveFirst, MoveLast, MoveNext, MovePrevious Methods

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthMoveFirstC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthMoveFirstX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthMoveFirstA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthMoveFirstS"}

Move to the first, last, next, or previous record in a specified **Recordset** object and make that record the current record.

Syntax

recordset.{**MoveFirst** | **MoveLast** | **MoveNext** | **MovePrevious**}

The *recordset* placeholder is an object variable representing an open **Recordset** object.

Remarks

Use the **MoveNext** method to move the current record position one record forward (towards 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.

Use the **MovePrevious** method to move the current record position one record backward (towards 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 "cached cursor" that will support backward cursor movement through the **Move** method. Because a cached cursor will load all records into memory, you should avoid caching more records than is necessary.

Use the **MoveFirst** method to move the current record position to the first record in the recordset. Using this method with a forward-only **Recordset** object generates an error.

Use the **MoveLast** method to move the current record position to the last record in the recordset. The **Recordset** object must support bookmarks; otherwise, the method call will generate an error.

NextRecordset Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthNextRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthNextRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthNextRecordsetA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthNextRecordsetS"}

Retrieves the next **Recordset**, if any, from a multiple recordset-returning query, and indicates whether one or more additional **Recordset** objects are pending.

Syntax

Boolean = *recordset*.**NextRecordset**

The **NextRecordset** method syntax has these parts:

Part	Description
Boolean	A **Boolean** variable that indicates if there are any further query results.
recordset	An object variable representing the **Recordset** object to which you want to return pending records.

Remarks

Use the **NextRecordset** method to test for the presence of additional data as the result of a compound query that returns more than one **Recordset** object.

Upon execution of a compound query, the results of the first query will be available in the **Recordset** object variable to which the **Execute** method assigns the query results. Using the **NextRecordset** method will allow you to obtain the recordsets from subsequent queries.

If *Boolean* is **True**, the results from the next part of the compound query are currently available in *recordset*. If *Boolean* is **False**, no more results are pending and *recordset* is now empty.

As long as there are still parts of the compound query whose results have not yet been examined, the **NextRecordset** method will return **True** until the last part of the compound query has been tested, even if some of the intervening recordsets are empty.

Open Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthOpenC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthOpenX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthOpenA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthOpenS"}

- **Connection** Opens a connection to a data source.
- **Recordset** Opens a cursor.

Syntax

For a **Connection** object:

connection.**Open** *ConnectionString*, *UserID*, *Password*

For a **Recordset** object:

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

The **Open** method syntax has these parts.

Part	Description
connection	An object variable representing an existing **Connection** object.
recordset	An object variable representing an existing **Recordset** object.
ConnectionString	Optional. A **String** containing connection information. See the **ConnectionString** property for details on valid settings.
UserID	Optional. A **String** containing a user name to use when establishing the connection.
Password	Optional. A **String** containing a password to use when establishing the connection.
Source	Optional. A **Variant** containing a valid **Command** object variable name, an SQL statement, a table name, or a stored procedure call.
ActiveConnection	Optional. A **Variant** containing a valid **Connection** object variable name or a **String** containing a definition for a connection.
CursorType	Optional. A **Long** expression that determines the type of cursor that the provider should use when opening the **Recordset**. Can be one of the following constants: • **adOpenForwardOnly**, 0 (Default) • **adOpenKeyset**, 1 • **adOpenDynamic**, 2 • **adOpenStatic**, 3 See the **CursorType** property for definitions of these settings.
LockType	Optional. A **Long** expression that determines what type of locking (concurrency) the provider should use when opening the **Recordset**. Can be one of the following constants: • **adConcurDefault**, -1 (Default) • **adConcurReadOnly**, 1

- **adConcurPessimistic**, 2
- **adConcurOptimistic**, 3
- **adConcurBatchOptimistic**, 4

See the **LockType** property for definitions of these settings.

Options — Optional. A **Long** expression that indicates how the provider should evaluate the *Source* argument if it represents something other than a **Command** object. Can be one of the following constants:

- **adOpenDefault**, 0 (Default) — The client application calls the provider to determine the nature of the *Source* argument.
- **adOpenSQLString**, 1 — The provider evaluates the *Source* argument as an SQL string.
- **adOpenStoredProc**, 2 — The provider evaluates the *Source* argument as a stored procedure.
- **adOpenTable**, 3 — The provider evaluates the *Source* argument as a table name.

Remarks

Use the **Open** method on a **Connection** object or a **Recordset** object to activate the object for use.

Connection

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 results.

Use the optional *ConnectionString* argument to specify a Data Source Name (DSN) or a detailed connection string containing a series of *parameter=value* arguments separated by semicolons. If the argument contains an equal sign ("="), ADO assumes that you are providing a connection string rather than a DSN. 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.

The provider determines whether the optional *UserID* and *Password* arguments override any user or password information provided in the *ConnectionString* parameter.

Recordset

Using the **Open** method on a **Recordset** object opens a cursor that represents records from a base table or the results of a query.

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, or a table name. If the *Source* argument is a **Command** object, specifying an *ActiveConnection* argument or an *Options* argument in the **Open** method generates an error.

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.

For the arguments that correspond directly to properties of a **Recordset** object (*Source*, *ActiveConnection*, *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 during 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 For **Recordset** objects whose **Source** property is set to a valid **Command** object, the **ActiveConnection** property is read-only, 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, 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 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.

Refresh Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthRefreshC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthRefreshX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthRefreshA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthRefreshS"}

Updates the objects in a collection to reflect objects available from and specific to the provider.

Syntax

collection.**Refresh**

The *collection* placeholder represents a **Parameters**, **Fields**, **Errors**, or **Properties** collection.

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 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.

Fields

Using the **Refresh** method on the **Fields** collection has no visible effect. To retrieve changes to the underlying database structure, you must use either the **Requery** method or, if the **Recordset** object does not support bookmarks, the **MoveFirst** method.

Errors

Using the **Refresh** method on the **Errors** collection has no visible effect as this collection is always current.

Properties

Using the **Refresh** method on a **Properties** collection of some objects populates the collection with the dynamic properties the provider exposes. These properties provide information about functionality specific to the provider beyond the built-in properties ADO supports.

Requery Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthRequeryC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthRequeryX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthRequeryA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthRequeryS"}

Updates the data in a **Recordset** object by re-executing the query on which the object is based.

Syntax

recordset.**Requery**

The *recordset* placeholder is an object variable representing an open **Recordset** object.

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.

Resync Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthResyncC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthResyncX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthResyncA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthResyncS"}

Refreshes the data in the current **Recordset** object from the underlying database.

Syntax

recordset.**Resync** *Affect*

The **Resync** method syntax has these parts:

Part	Description
recordset	An object variable representing an open **Recordset** object.
Affect	Optional. A **Long** expression that determines on how many records the **Resync** method should act. Can be one of the following constants: • **adAffectCurrent**, 1 — Retrieves underlying values only for the current record. • **adAffectAll**, 2 (Default) — Retrieves underlying values for all the records in the **Recordset** object. • **adAffectGroupFilter**, ??? — Retrieves underlying values for the records that satisfy the current **Filter** property setting. There must be a valid **Filter** on the **Recordset** in order to use this constant.

Remarks

Use the **Resync** method to re-synchronize 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. Calling the **Resync** method cancels any pending batch updates.

Unlike the **Refresh** 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 but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records.

Supports Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthSupportsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthSupportsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthSupportsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthSupportsS"}

Determines whether a specified **Recordset** object supports a particular type of functionality.

Syntax

Boolean = *recordset*.**Supports**(*CursorOptions*)

The **Supports** method syntax has these parts:

Part	Description
Boolean	A **Boolean** variable that indicates whether the specified functionality is available.
recordset	An object variable representing an open **Recordset** object.
CursorOptions	A **Long** expression that consists of one or more of the constants listed under Remarks.

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**.

The *CursorOptions* argument can be set to any combination of the following:

Constant	Value	Functionality
adBookmark	8192	You can use the **Bookmark** property to access specific records.
adDelete	16779264	You can use the **Delete** method to delete records.
adHoldRecords	256	You can retrieve more records or change the next retrieve position without committing all pending changes and releasing all currently held records.
adMovePrevious	512	You can use the **MovePrevious** or **Move** methods to move the current record position backward.
adResync	131072	You can update the cursor with the data visible in the underlying database.
adApproxPosition	16384	You can retrieve records from positions based on bookmarks or approximate positions.
adAddNew	16778240	You can use the **AddNew** method to add new records.
adUpdate	16809984	You can use the **Edit** method to modify existing data.
adUpdateBatch	65536	You can use batch updating to transmit changes to the provider in groups.

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 or not the provider can support the specified functionality assuming that 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 multi-table join, some of the columns of which are not updatable.

Update Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthUpdateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthUpdateX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthUpdateA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthUpdateS"}

Saves any changes you make to the current record of a **Recordset** object.

Syntax

recordset.**Update** *Fields*, *Values*

The **Update** method syntax has these parts:

Part	Description
recordset	An object variable representing an open, updatable **Recordset** object.
Fields	Optional. A **Variant** representing a single name or a **Variant** array representing names of the field(s).
Values	Optional. A **Variant** representing a single value or a **Variant** array representing values for the field(s) in the new record.

Remarks

Use the **Update** method to save any changes you make to the current record of a **Recordset** object since calling the **AddNew** or **Edit** method. 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, then 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 to discard a newly added record.

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

UpdateBatch Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthUpdateBatchC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthUpdateBatchX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthUpdateBatchA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthUpdateBatchS"}

Writes all pending batch updates to disk.

Syntax

recordset.**UpdateBatch** *Affect*

The **UpdateBatch** method syntax has these parts:

Part	Description
recordset	An object variable representing an open **Recordset** object.
Affect	Optional. A **Long** expression that determines on how many records the **UpdateBatch** method should act. Can be one of the following constants: • **adAffectCurrent**, 1 — Transmit changes only for the current record. • **adAffectAll**, 2 (Default) — Transmit changes for all the records in the **Recordset** object. • **adAffectGroupFilter**, ??? — Transmit changes for records that satisfy the current **Filter** property setting. There must be a valid **Filter** on the **Recordset** in order to use this constant.

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, then 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 the attempt to transmit changes 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.

To cancel all pending batch updates, use the **CancelUpdateBatch** method.

AbsolutePage Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproAbsolutePageC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproAbsolutePageX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproAbsolutePageA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproAbsolutePageS"}

Indicates the absolute "page" on which the current record is located.

Settings and Return Values

Sets or returns a **Long** value from 1 to the number of pages in the **Recordset** object (**PageCount**). The following values are also valid:

Constant	Value	Description
adAbPosUnknown	-1	No current record
adAbPosBOF	-2	Before the first record
adAbPosEOF	-3	After the last record

Remarks

Use the **AbsolutePage** property to set or return the "page" number on which the current record is located. Use the **PageSize** property to logically divide the **Recordset** object into a series of pages, each of which has the number of records equal to **PageSize**.

Like the **AbsolutePosition** property, **AbsolutePage** is 1-based and returns 1 when the current record is the first record in the **Recordset**. If the provider cannot determine the absolute page number, it returns **adAbPosUnknown** (-1). If either the **BOF** or **EOF** property is **True**, the provider returns the constants **adAbPosBOF** or **adAbPosEOF** respectively.

You can set this property to move to the first record of a particular page. You can obtain the total number of pages from the **PageCount** property.

AbsolutePosition Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproAbsolutePositionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproAbsolutePositionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproAbsolutePositionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproAbsolutePositionS"}

Indicates the ordinal position of a **Recordset** object's current record.

Settings and Return Values

Sets or returns a **Long** from 1 to the number of records in the **Recordset** object (**RecordCount**). The following values are also valid:

Constant	Value	Description
adAbPosUnknown	-1	No current record
adAbPosBOF	-2	Before the first record
adAbPosEOF	-3	After the last record

Remarks

Use the **AbsolutePosition** property to move to a record based on its ordinal position in the **Recordset** object or to return the ordinal position of the current record.

Like the **AbsolutePage** property, **AbsolutePosition** is 1-based and returns 1 when the current record is the first record in the **Recordset**. If the **Recordset** is empty or the provider cannot determine the absolute position of the current record, the property value is **adAbPosUnknown** (-1). If either the **BOF** or **EOF** property is **True**, the provider returns the constants **adAbPosBOF** or **adAbPosEOF** respectively.

When you set the **AbsolutePosition** property to a record in the current cache, ADO moves the current record position but does not reload the data in the cache from the provider. When you set the **AbsolutePosition** property to a record outside the current cache, ADO retrieves a new group of records starting with the record you specified. The **CacheSize** property determines the size of this group.

You can obtain the total number of records in the **Recordset** object from the **RecordCount** property. When you add new records, they appear at the end of the **Recordset** and receive the next largest **AbsolutePosition** values. Reading the **AbsolutePosition** property of a deleted record generates an error.

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(s). There is also no assurance that a given record will have the same **AbsolutePosition** if the **Recordset** object is requeried 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.

ActiveConnection Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproActiveConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproActiveConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproActiveConnectionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproActiveConnectionS"}

Indicates to which **Connection** object the specified **Command** or **Recordset** object currently belongs.

Settings and Return Values

Sets or returns a **String** containing the definition for a connection or a **Connection** object variable. Default is **Null**.

Remarks

Use the **ActiveConnection** property to set or return an open **Connection** object over which the specified **Command** object should execute or the specified **Recordset** should be opened. If you pass a connection definition for this argument, ADO opens a new connection using the specified parameters.

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.

Setting this property to **Null** dis-associates the **Command** object from the current **Connection**, clears the **Parameters** collection of the **Command** object, 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. If this property is currently set to an open **Connection** object and you change the setting to another open **Connection** object, the **Parameters** collection will remain intact.

Closing the **Connection** object with which a **Command** object is associated sets the **ActiveConnection** property to **Null**. 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 definition 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.

Usually, you would set this property on a closed **Recordset** object and then open the **Recordset** object. If you are using batch updating and change the **ActiveConnection** property setting of an open **Recordset** object to another open **Connection** object, you can post any batched updates in the **Recordset** object to the underlying database of the new **Connection** assuming that the schemas of the two databases match.

ActualSize Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproActualSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproActualSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproActualSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproActualSizeS"}

Indicates the actual length of a field's value.

Return Values

Returns a **Long** value.

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.

The **ActualSize** and **DefinedSize** properties are different as shown in the following example: a **Field** object with a declared type of **adChar** 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.

Note To get the **ActualSize** value of a long binary field, the **Recordset** object must support bookmarks.

Attributes Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproAttributesC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproAttributesX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproAttributesA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproAttributesS"}

Indicates one or more characteristics of an object.

Settings and Return Values

Sets or returns a **Long** value.

For a **Connection** object, the value can be any one or more of these constants (default is zero):

Constant	Value	Description
adXactPollAsync	2	Performs asynchronous commits. You will not be able to confirm the outcome of transactions you commit.
adXactPollSyncPhaseOne	4	Calls commits after phase one of the two-phase commit protocol.
adXactCommitRetaining	131072	Performs retaining commits, that is, calling **CommitTrans** automatically starts a new transaction. Not all providers will support this.
adXactAbortRetaining	262144	Performs retaining aborts, that is, calling **RollbackTrans** automatically starts a new transaction. Not all providers will support this.
adXactAbortAsync	524288	Performs asynchronous aborts. You will not be able to confirm the outcome of transactions you rollback.

For a **Parameter** object, the value can be any one or more of these constants:

Constant	Value	Description
adParamSigned	16	Indicates that the parameter accepts signed values. (Default.)
adParamNullable	64	Indicates that the parameter accepts **Null** values.
adParamLong	128	Indicates that the parameter accepts long binary data.

For a **Field** object, the value specifies characteristics of the field and can be a sum of any one or more of these constants:

Constant	Value	Description
adFldBookmark	1	Indicates that the field contains a bookmark.
adFldMayDefer	2	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.

adFldUpdatable	4	Indicates that you can write to the field.
adFldUnknownUpdatable	8	Indicates that the provider cannot determine if you can write to the field.
adFldFixed	16	Indicates that the field contains fixed-length data.
adFldIsNullable	32	Indicates that the field accepts **Null** values.
adFldMayBeNull	64	Indicates that you can read **Null** values from the field.
adFldLong	128	Indicates that the field is a long binary field. Also indicates that you can use the **AppendChunk** and **GetChunk** methods.
adFldRowID	256	Indicates that the field contains some kind of record ID (record number, unique identifier, and so forth).
adFldRowVersion	512	Indicates that the field contains some kind of time or date stamp used to track updates.
adFldCacheDeferred	4096	Indicates that the provider caches field values and that subsequent reads are done from the cache.

For a **Property** object, the value can be any one or more of these constants:

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.

Remarks

Use the **Attributes** property to set or return characteristics of **Connection** objects, **Parameter** objects, **Field** objects, or **Property** objects.

For **Connection** objects, the **Attributes** property is read/write. Also, the **adXactPollAsync** and **adXactPollSyncPhaseOne** constants are mutually exclusive; adding both constants to the **Attributes** property generates an error.

For **Parameter** objects, the **Attributes** property is read/write. For **Field** and **Property** objects, the **Attributes** property is read-only.

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 (for example, the **adXactPollAsync** and **adXactPollSyncPhaseOne** constants on the **Connection** object).

BOF, EOF Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproBOFC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproBOFX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproBOFA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproBOFS"}

- **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 Values

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**, and the **Recordset** object's **RecordCount** property setting is zero. 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.

Calling the **Delete** method, even if it removes the only remaining record from a **Recordset**, won't change the setting of the **BOF** or **EOF** property.

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**

Bookmark Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproBookmarkC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproBookmarkX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproBookmarkA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproBookmarkS"}

Returns 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. (Data type is **Variant** array of **Byte** data.)

Remarks

Use the **Bookmark** property to save the position of the current record and return to that record at any time. Bookmarks are only available 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 can't use bookmarks from different **Recordset** objects interchangeably, even if they were created from the same source or command.

CacheSize Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCacheSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCacheSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCacheSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCacheSizeS"}

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 which must be greater than 0. Default is 1 for forward-only cursors, 10 for all other types.

Remarks

Use the **CacheSize** property to control how many records the provider keeps in its buffer and how many to retrieve at one time into local memory. 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.

The value of this property 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; 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.

CommandText Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCommandTextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCommandTextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCommandTextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCommandTextS"}

Contains the text of a command that you want to issue against a provider.

Settings and Return Values

Sets or returns a **String** value containing an SQL statement, a table name or a stored procedure call. Default is "" (zero-length string).

Remarks

Use the **CommandText** property to set or return the text version of the query in a **Command** object.

The contents of the **CommandText** property are specific to the provider and can be standard SQL syntax or any special command format that the provider supports.

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 when you call the **Execute** or **Open** methods. Depending on the **CommandType** property setting, ADO may alter the **CommandText** property during preparation or execution. After you call the **Execute** method, you may read the **CommandText** property to see if its value has changed.

CommandTimeout Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCommandTimeoutC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCommandTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCommandTimeoutA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCommandTimeoutS"}

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 a command due to delays from network traffic or heavy server use. If the time from the **CommandTimeout** property setting elapses prior to execution of the command, 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 supports the **CommandTimeout** functionality.

For **Connection** objects, the **CommandTimeout** property is read/write.

When you use **CommandTimeout** on a **Connection** object, you set a global value for all commands executed and all recordsets opened on that connection. You can override this value for a specific command by setting the **CommandTimeout** property of the appropriate **Command** object.

CommandType Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCommandTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCommandTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCommandTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCommandTypeS"}

Indicates the type of a **Command** object.

Settings and Return Values

Sets or returns a **Long** value equal to one of the following constants:

Constant	Value	Description
adCmdDefault	-1	???
adCmdUnknown	0	The type of command in the **CommandText** property is not known. (Default)
adCmdText	1	Evaluates **CommandText** as a textual definition of a command.
adCmdTable	2	Evaluates **CommandText** as a table name.
adCmdStoredProc	4	Evaluates **CommandText** as a stored procedure.

Remarks

Use the **CommandType** property to optimize evaluation of the **CommandText** property. If the **CommandType** property value equals **adCmdUnknown** (0), 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.

ConnectionString Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproConnectionStringC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproConnectionStringX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproConnectionStringA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproConnectionStringS"}

****** THIS TOPIC STILL IN PROGRESS. ******

Contains connection parameters.

Settings and Return Values

Sets or returns a **String** value.

Remarks

This property reflects the connection string or information used to connect to the data source. You can set this property while the connection is not actually open but it becomes read-only once the **Open** method has succeeded. This property becomes read/write again after the **Close** method has succeeded.

This property is set to the actual connection string that was used in creating the connection that gets returned.

The behavior matches that of DAO 3.5 with respect to passwords.

ConnectionTimeout Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproConnectionTimeoutC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproConnectionTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproConnectionTimeoutA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproConnectionTimeoutS"}

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.

Count Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCountS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCountA"}

Indicates the number of objects in a collection.

Return Values

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 one. If you 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.

CursorType Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCursorTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCusorTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCursorTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCursorTypeS"}

****** THIS TOPIC STILL IN PROGRESS. ******

Indicates the type of cursor used in a **Recordset** object.

Settings and Return Values

Sets or returns a **Long** value equal to one of the following constants:

Constant	Value	Description
adOpenDefault	-1	If the provider is unable to open the cursor you specify, it may open a different type and set the **CursorType** property accordingly. If the cursor used by the provider cannot be determined, **CursorType** is set to this value.
adOpenForwardOnly	0	Forward-only cursor. Identical to a static cursor except that you can only scroll forward through records. This improves performance in situations when you only need to make a single pass through a recordset. (Default)
adOpenKeyset	1	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.
adOpenDynamic	2	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.
adOpenStatic	3	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.

Remarks

Use the **CursorType** property to determine the type of cursor that should be used when opening the **Recordset** object. The **CursorType** property is read/write when the recordset is closed and read-only when it is open.

If a provider does not support the requested cursor type, the provider may return another cursor type. To verify specific functionality of the returned cursor, use the **Supports** method.

****** TO BE INCLUDED: A chart correlating cursor types to Supports method features. ******

DefaultDatabase Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDefaultDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDefaultDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDefaultDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDefaultDatabaseS"}

Indicates the default database for a **Connection** object.

Settings and Return Values

Sets or returns a **String** 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.

DefinedSize Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDefinedSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDefinedSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDefinedSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDefinedSizeS"}

Indicates the defined size of a **Field** object.

Return Values

Returns a **Variant** 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 as shown in the following example: a **Field** object with a declared type of **adChar** 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 up to 50.

Description Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDescriptionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDescriptionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDescriptionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDescriptionS"}

A descriptive string associated with an error.

Return Values

Returns a **String** value.

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 it receives. Enumerate the **Errors** collection to trace the errors that the provider passes.

Direction Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDirectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDirectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDirectionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDirectionS"}

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.

Settings and Return Values

Sets or returns a **Long** value equal to one of the following constants:

Constant	Value	Description
adParamInput	1	Input parameter (Default)
adParamOutput	2	Output parameter
adParamInputOutput	3	Input and output parameter
adParamReturnValue	4	Return 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 prior to executing the query.

EditMode Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproEditModeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproEditModeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproEditModeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproEditModeS"}

Indicates the editing status of the current record.

Return Values

Returns a **Long** value equal to one of the following constants:

Constant	Value	Description
adEditNone	0	No editing operation is in progress.
adEditInProgress	1	Data in the current record has been modified but not yet saved.
adEditAdd	2	The **AddNew** method has been invoked, and the current record in the copy buffer is a new record that hasn't been saved in the database.

Remarks

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.

Filter Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproFilterC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproFilterX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproFilterS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproFilterA"}

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`.
- One of the following constants:

Constant	Value	Description
adFilterNone	0	Removes the current filter and restores all records to view.
adFilterPendingRecords	1	Allows you to view only records that have changed but not yet sent to the server. Only applicable for batch update mode.
adFilterAffectedRecords	2	Allows you to view only records affected by the last **Delete**, **Resync**, **UpdateBatch**, or **CancelUpdateBatch** call.
adFilterFetchedRecords	3	Allows you to view only records that have already been retrieved.
adFilterPredicate	4	???

- Array of bookmarks — an array of unique bookmark values that point to records in the **Recordset** object.

Remarks

Use the **Filter** property to selectively screen out records in a **Recordset** object. The filtered **Recordset** becomes the current cursor. This affects other properties such as **AbsolutePosition**, **AbsolutePage**, **RecordCount**, and **PageCount** that return values based on the current cursor.

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` or `OR` (for example, "`LastName = 'Smith' AND FirstName = 'John'`"). 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: `<`, `>`, `<=`, `>=`, `<>`, `=`, `Like`, `Not`.
- *Value* is the value with which you will compare the field values (for example, `'Smith'`, `#8/24/95#`, `12.345` or `$50.00`). If *Operator* is `Like`, *Value* can use wildcards. Use single quotes with strings and pound signs (`#`) with dates. For numbers, you can use decimal points, dollar signs, and scientific notation.

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 to a zero-length string ("") has the same effect as using the **adFilterNone** constant.

See the **Bookmark** property for an explanation of bookmark values from which you can build an array to use with the **Filter** property.

HelpContext, HelpFile Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproHelpContextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproHelpContextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproHelpContextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproHelpContextS"}

Indicates the help file and topic associated with an error.

Return Values

- **HelpContextID** — returns a context ID, as a **Long** value, for a topic in a Microsoft Windows Help file.
- **HelpFile** — returns a **String** that evaluates to a fully qualified 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 ("").

IsolationLevel Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproIsolationLevelC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproIsolationLevelX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproIsolationLevelA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproIsolationLevelS"}

Indicates the level of isolation for a **Connection** object.

Settings and Return Values

Sets or returns a **Long** equal to one of the following constants:

Constant	Value	Description
adXactUnspecified	-1	If the provider is using a different **IsolationLevel** than specified but which one cannot be determined, the property returns this value.
adXactChaos	16	???
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've been committed. (Default.)
adXactReadCommitted	4096	Same as **adXactCursorStability**.
adXactRepeatableRead	65536	Indicates that from one transaction you cannot see changes made in other transactions, but that requerying can bring new recordsets.
adXactIsolated	1048576	Indicates that transactions are conducted in isolation of other transactions.
adXactSerializable	1048576	Same as **adXactIsolated**.

Remarks

Use the **IsolationLevel** property to set the isolation level of a **Connection** object. The **IsolationLevel** property is read/write. The setting does not take effect until the next time you call the **BeginTrans** method.

LockType Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproLockTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproLockTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproLockTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproLockTypeS"}

Indicates the type of locks placed on records during editing.

Settings and Return Values

Sets or returns a **Long** value equal to one of the following constants:

Constant	Value	Description
adLockDefault	-1	Provider determines lock type (typically read-only).
adLockReadOnly	1	Read-only — you cannot alter the data.
adLockPessimistic	2	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 upon editing.
adLockOptimistic	3	Optimistic locking, record by record — the provider uses optimistic locking, locking records only when you call the **Update** method.
adLockBatchOptimistic	4	Optimistic batch updates — required for batch update mode as opposed to immediate update mode.

Remarks

Use the **LockType** property to determine what type of locking the provider should use when opening a **Recordset** object or to return the type of locking in use on an open **Recordset** object. The **LockType** property is read/write when the **Recordset** is closed and read-only when it is open.

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.

MaxRecords Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproMaxRecordsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproMaxRecordsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproMaxRecordsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproMaxRecordsS"}

Indicates the maximum number of records to return to a **Recordset** from a query.

Settings and Return Values

Sets or returns a **Long** value. Default is zero.

Remarks

Use the **MaxRecords** property to limit the number of records the provider returns from the data source. The default setting of this property is zero which means that the provider returns all requested records. The **MaxRecords** property is read/write when the **Recordset** is closed and read-only when it is open.

Mode Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproModeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproModeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproModeA"} ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproModeS"}

****** THIS TOPIC STILL IN PROGRESS. ******

Indicates the available permissions for modifying data in a **Connection**.

Settings and Return Values

Sets or returns a **Long** value that can be one of the following constants:

Constant	Value	Description
adModeDefault	0	(Default)
adModeRead	1	
adModeWrite	2	
adModeReadWrite	3	
adModeShareDenyRead	4	
adModeShareDenyWrite	8	
adModeShareExclusive	12	
adModeShareDenyNone	16	

Remarks

Use the **Mode** property to set or return the access permissions in use by the provider on the current connection. The **Mode** property is read/write when the connection is closed and read-only when it is open.

Name Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproNameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproNameS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproNameA"}

Indicates the name for a **Field**, **Parameter,** or **Property** object.

Settings and Return Values

Sets or returns a **String** value.

Remarks

Use the **Name** property to refer to a **Field**, **Parameter**, or **Property** object within its respective 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")`.

NativeError Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproNativeErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproNativeErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproNativeErrorA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproNativeErrorS"}

Indicates the provider-specific error code for a given **Error** object.

Return Values

Returns a **Long** value.

Remarks

Use the **NativeError** property to retrieve the provider-specific error code for a particular **Error** object.

Number Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproNumberC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproNumberX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproNumberA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproNumberS"}

Indicates the number that uniquely identifies an error.

Return Values

Returns a **Long** value.

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.

OriginalValue Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproOriginalValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproOriginalValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproOriginalValueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproOriginalValueS"}

Indicates the value of a **Field** that existed in the record before any changes were made, or in the database when the last batch update began.

Return Values

Returns a **Variant** value.

Remarks

Use the **OriginalValue** property to return the original field value for a field from the current record.

In immediate update mode, the **OriginalValue** property returns the field value that existed before you called any methods that set the field value, but 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, the **OriginalValue** property returns the field value that existed before any methods that set the value were called on the field, but since the last **UpdateBatch**. This is the same value that the **CancelUpdateBatch** 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.

PageCount Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPageCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPageCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPageCountA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPageCountS"}

Indicates how many "pages" of data the **Recordset** object contains.

Return Values

Returns a **Long** value.

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. 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.

PageSize Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPageSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPageSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPageSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPageSizeS"}

Indicates how many records constitute one "page" in the **Recordset**.

Settings and Return Values

Sets or returns a **Long** value, indicating how many records are on a page. 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 where the first record of a particular page is.

Precision Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPrecisionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPrecisionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPrecisionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPrecisionS"}

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, indicating the maximum total 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.

For **Parameter** objects, the **Precision** property is read/write. For **Field** objects, the **Precision** property is read-only.

Prepared Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPreparedC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPreparedX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPreparedA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPreparedS"}

Indicates whether or not to create a prepared statement from the command before execution.

Settings and Return Values

Sets or returns a **Boolean** value.

Remarks

Use the **Prepared** property to have the provider create a temporary stored representation of the query specified in the **CommandText** property before a **Command** object's first execution. Setting this property to **True** requests the provider to compile a command on its first execution. This may slow a command's first execution, but once the provider compiles a command, it 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 ignores any requests to prepare the command and sets the **Prepared** property to **False**.

Provider Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproProviderC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproProviderX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproProviderA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproProviderS"}

Indicates the name of the provider for a **Connection** object.

Settings and Return Values

Sets or returns a **String** value.

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 **ConnectionString** property or the *ConnectionString* argument of the **Open** method. If no provider is specified, the property will default to MSDASQL.

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 invalid, an error occurs.

RecordCount Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproRecordCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproRecordCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproRecordCountA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproRecordCountS"}

Indicates the current number of records in a **Recordset** object.

Return Values

Returns a **Long** value.

Remarks

Use the **RecordCount** property to find out how many records are in a **Recordset** object. The property returns **adUnknown** when ADO cannot determine the number of records. Reading the **RecordCount** property on a closed **Recordset** causes an error.

If the **Recordset** object supports approximate positioning or bookmarks, 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.

Scale Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproScaleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproScaleX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproScaleS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproScaleA"}

Indicates the scale of **Numeric** values in a **Parameter** or **Field** object.

Settings and Return Values

Sets or returns a **Byte** value, indicating the number of decimal places to which numeric values will be resolved.

Remarks

Use the **Scale** 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 **Scale** property is read/write. For **Field** objects, the **Scale** property is read-only.

Size Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSizeX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSizeS"}

Indicates the maximum size, in bytes, of a **Parameter** object.

Settings and Return Values

Sets or returns a **Long** value that indicates the maximum size in bytes 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. The **Size** property is read/write.

Source Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSourceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSourceX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSourceA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSourceS"}

- **Recordset** — indicates the source for the data in the **Recordset** object (**Command** object, SQL statement, table name, or stored procedure).
- **Error** — indicates the name of the object or application that originally generated the error.

Settings and Return Values

- **Recordset** — sets or returns a **String** value or **Command** object reference.
- **Error** — returns a **String** value.

Remarks

Use the **Source** property on a **Recordset** object to determine from where the **Recordset** object's data comes, or on an **Error** object to create error-handling routines.

Recordset

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. The **Source** property is read/write for closed **Recordset** objects and read-only for open **Recordset** objects.

If the **Source** property specifies a **Command** object, the **ActiveConnection** property of the **Recordset** object will inherit the value of the **ActiveConnection** property for the specified **Command** object. 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.

Error

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 ADO, this property is **ADO.***ObjectName* where *ObjectName* is the name of the object that triggered the error. The **Source** property is read-only for **Error** objects.

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.

SQLState Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSQLStateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSQLStateX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSQLStateA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSQLStateS"}

Indicates the SQL state for a given **Error** object.

Return Values

Returns a five-character **String** that follows the ANSI SQL standard.

Remarks

Use the **SQLState** property to obtain the SQL state of an Error object expressed as a five-character **String** that follows the ANSI SQL standard.

Status Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproStatusC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproStatusX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproStatusS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproStatusA"}

****** THIS TOPIC STILL IN PROGRESS. ******

Indicates the results of the last **Update** or **UpdateBatch** call on a **Recordset** object.

Return Values

Returns a **Long** value made up of a combination of the following constants:

Constant	Value	Description
adRecOK	0	The record was succesfully updated.
adRecNew	1	The record is new.
adRecModified	2	The record was modified.
adRecDeleted	4	The record was deleted.
adRecUnmodified	8	The record was not modified.
adRecInvalid	16	The record was not saved because its bookmark is invalid.
adRecMultipleChanges	64	The record was not saved because it would have affected multiple records.
adRecPendingChanges	128	The record was not saved because it refers to a pending insert.
adRecCanceled	256	The record was not saved because the operation was canceled.
adRecCantRelease	1024	The new record was not saved because of existing record locks.
adRecConcurrencyViolation	2048	The record was not saved because optimistic concurrency was in use.
adRecIntegrityViolation	4096	The record was not saved because the user violated integrity constraints.
adRecMaxChangesExceeded	8192	The record was not saved because there were too many pending changes.
adRecObjectOpen	16384	The record was not saved because of a conflict with an open storage object.
adRecOutOfMemory	32768	The record was not saved because the computer has run out of memory.
adRecPermissionDenied	65536	The record was not saved because the user has insufficient permissions.
adRecSchemaViolation	131072	The record was not saved

		because it violates the structure of the underlying database.
adRecDBDeleted	262144	The record has already been deleted from the data source.

Remarks

Use the **Status** property to view the status of records that fail during bulk operations such as when you call the **Delete**, **Resync**, **UpdateBatch** or **CancelUpdateBatch** methods on a **Recordset** object or set the **Filter** property on a **Recordset** object. With this property, you can determine how a given record failed and resolve it accordingly.

Type Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproTypeS"}

Indicates the operational type or data type of a **Parameter**, **Field**, or **Property** object.

Settings and Return Values

Sets or returns a **Long** value equal to one of the following constants:

Constant	Value	Description
adBigInt	20	An eight-byte signed integer.
adBinary	128	A binary value.
adBoolean	11	A **Boolean** value.
adBSTR	8	A null-terminated character string (Unicode).
adChar	129	A **String** value.
adCurrency	6	A currency value (eight-byte signed integer scaled by 10,000).
adDate	7	A **Date** value.
adDBDate	133	A date value (*yyyymmdd*).
adDBTime	134	A time value (*hhmmss*).
adDBTimeStamp	135	A date-time stamp (*yyyymmddhhmmss* plus a fraction in billionths).
adDecimal	14	An exact numeric value with a fixed precision and scale.
adDefault	-1	???
adDouble	5	A double-precision floating point value.
adEmpty	0	No value was specified.
adError	10	A 32-bit error code.
adGUID	72	A globally unique identifier (GUID).
adIDispatch	9	A pointer to an **IDispatch** interface on an OLE object.
adInteger	3	A four-byte signed integer.
adIUnknown	13	A pointer to an **IUnknown** interface on an OLE object.
adLongVarBinary	205	A long binary value. (**Parameter** object only.)
adLongVarChar	201	A long **String** value. (**Parameter** object only.)
adLongVarWChar	203	A long null-terminated string value. (**Parameter** object only.)
adNumeric	131	An exact numeric value with a fixed precision and scale.
adSingle	4	A single-precision floating point value.
adSmallInt	2	A two-byte signed integer.
adTinyInt	16	A one-byte signed integer.
adUnsignedBigInt	21	An eight-byte unsigned integer.
adUnsignedInt	19	A four-byte unsigned integer.

adUnsignedSmallInt	18	A two-byte unsigned integer.
adUnsignedTinyInt	17	A one-byte unsigned integer.
adUserDefined	132	A user-defined variable.
adVarBinary	204	A binary value. (**Parameter** object only.)
adVarChar	200	A **String** value. (**Parameter** object only.)
adVariant	12	An OLE Automation **Variant**.
adVarWChar	202	A null-terminated Unicode character string. (**Parameter** object only.)
adWChar	130	A null-terminated Unicode character string.

Remarks

For **Parameter** objects, the **Type** property is read/write. For all other objects, the **Type** property is read-only.

UnderlyingValue Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproUnderlyingValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproUnderlyingValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproUnderlyingValueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproUnderlyingValueS"}

Indicates a **Field** object's current value in the database.

Return Values

Returns a **Variant** value.

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 only returns 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.

Value Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproValueS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproValueA"}

Indicates the value assigned to a **Field**, **Parameter**, or **Property** object.

Settings and Return Values

Sets or returns a **Variant** value. 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 topics for the respective objects for more information.

ADO allows setting and returning long binary data with the **Value** property.

Version Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproVersionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproVersionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproVersionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproVersionS"}

Indicates the ADO version number.

Return Values

Returns a **String** value.

Remarks

Use the **Version** property to return the version number of the ADO implementation.

You can use this property to ensure that you are programming to the expected version of the provider, or to write conditional code based upon the version of the library. For instance, if a provider contained a bug for which you wrote special code, but the bug was fixed in a later version, you could check the version property to determine if the special code was still necessary.

The version of the actual data source may be exposed as a dynamic property in the **Properties** collection.