The ADO 2.0 Object Model

Overview
ActiveX® Data Objects (ADO) provides a common programming model for any OLE DB data source; it is essentially a collection of objects that expose the attributes and methods used to communicate with a data source. ADO uses general OLE DB providers to access unique features of specific data sources; it also uses native OLE DB providers, including a specific OLE DB provider that provides access to Open Database Connectivity (ODBC) drivers. Designed to replace the need for all other high-level data access methods, ADO can access relational, Indexed Sequential Access Method (ISAM), or hierarchical databases, or any type of data source—as long as there is an ODBC-compliant driver.

What ADO cannot do, however, is provide remote data to the client. Once the data has been retrieved and sent to the browser, the user cannot easily manipulate it or make changes to it within the client application. Data operations—including filtering and record modifications—must take place on the server, where the actual data manipulation objects reside.

How to Reference ADO 2.0 in Visual Basic & Access2000
To gain access to the ADO 2.0 objects in Visual Basic, set a reference to the appropriate ADO type library. There are two ADO type libraries. One is called ADODB and is contained in MSADO15.DLL. It appears in the References dialog box (available from the Project menu) as "Microsoft ActiveX Data Objects 2.0 Library". The other is called ADOR and is contained in MSADOR15.DLL. It appears in the References dialog as "Microsoft ActiveX Data Objects Recordset 2.0 Library". The first type library (ADODB) is the larger and more feature-rich of the two; it contains the main ADO objects and is the one you'll probably want to use in most circumstances. The second is a "lightweight" subset of the ADODB type library that supports only recordsets. You may choose to use that library instead if you plan to manipulate only recordsets.

( NOTE: You must be using the Visual Basic Editor to add the reference to ADODB. Follow these steps:
Tools | References | Check “Microsoft ActiveX Data Objects X.Y Library” where X.Y is the latest release registered in your machine).

Why Use ADO? (According to Microsoft!)
ADO is designed as an easy-to-use application level interface to Microsoft's newest and most powerful data access paradigm, OLE DB. OLE DB provides high-performance access to any data source, including relational and non-relational databases, email and file systems, text and graphics, custom business objects, and more. ADO is implemented for minimal network traffic in key Internet scenarios, and a minimal number of layers between the front-end and data source — all to provide a lightweight, high-performance interface.

Components of the ADO 2.0 Object Model
The ADO 2.0 object model is composed of eight objects.

Note All objects followed by an asterisk (*) are objects that also apply to the ADO Recordset type library (ADOR).

• Command object Contains information about a command, such as a query string, parameter definition, and so forth. The Command object is similar in functionality to RDO's rdoQuery object.

• Connection object Contains information about a data provider. The Connection object is similar in functionality to RDO's rdoConnection object, and it contains the information on schema. It also contains some of the functionality of the RDOEnvironment object, such as transaction control.
• **Error object** Contains extended information when an error occurs with a data provider. The Error object is similar in functionality to RDO's rdoError object. In comparison to RDO, however, the Errors collection is on the Connection object, whereas the rdoErrors collection is on the rdoEngine object in RDO 2.0.

• **Field object** Contains information about a single column of data in a recordset. The Field object is similar in functionality to RDO's rdoColumn object.

• **Parameter object** Contains a single parameter for a parameterized Command object. The Command object has a Parameters collection to contain all of its Parameter objects. The Parameter object is similar in functionality to RDO's rdoParameter object.

• **Property object** Contains a provider-defined characteristic of an ADO object. There is no RDO equivalent to this object, but DAO has a similar object. ADO objects can have two kinds of properties:
  - *Built-In Properties*: Properties which are "native" to ADO. That is, properties in ADO that are immediately available to any new object using the familiar MyObject.Property syntax. Built-in properties do not appear as Property objects in an object’s Properties collection, so while you can change their values, you can’t modify their characteristics or delete them.
  - *Dynamic properties*: Properties which are not native to ADO and are defined by the underlying data provider. They appear in the Properties collection of the appropriate ADO object. For example, a property specific to the data provider may indicate if a Recordset object supports transactions or updating. These additional properties appear as Property objects in the Recordset’s Properties collection. Dynamic properties can be referred to only through the collection using the MyObject.Properties(0) or MyObject.Properties("Name") syntax. Different data providers may implement one or more special properties to deal with special provider-specific operations.

• **Recordset object** The Recordset object contains records returned from a query as well as a cursor into those records. The Recordset object is similar in functionality to RDO’s rdoResultset object. You can open a Recordset (for example, perform a query) without explicitly opening a Connection object. If, however, you choose to create a Connection object, you can open multiple Recordset objects on the same connection.
Connection Object

A Connection object represents an open connection to a data source.

Remarks

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

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

- Configure the connection before opening it with the ConnectionString, ConnectionTimeout, and Mode properties.

- Set the CursorLocation property to invoke the Client Cursor Provider, which supports batch updates.

- Set the default database for the connection with the DefaultDatabase property.

- Set the level of isolation for the transactions opened on the connection with the IsolationLevel property.

- Specify an OLE DB provider with the Provider property.

- Establish, and later break, the physical connection to the data source with the Open and Close methods.

- Execute a command on the connection with the Execute method and configure the execution with the CommandTimeout property.

- 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.

- Obtain schema information about your database with the OpenSchema method.

Note To execute a query without using a Command object, pass a query string to the Execute method of a Connection object. However, a Command object is required when you want to persist the command text and re-execute it, or use query parameters.
You can create Connection objects independently of any other previously defined object.

**Note** You can execute commands or stored procedures as if they were native methods on the Connection object.

To execute a command, give the command a name using the Command object Name property. Set the Command object's ActiveConnection property to the connection. Then issue a statement where the command name is used as if it were a method on the Connection object, followed by any parameters, followed by a Recordset object if any rows are returned. Set the Recordset properties to customize the resulting recordset. For example:

```vba
Dim cnnCompany As New ADODB.Connection
Dim cmd As New ADODB.Command
Dim rst As New ADODB.Recordset
...
cnnCompany.Open "...
cmd.Name = "yourCommandName"
cmd.ActiveConnection = cnnCompany
...
'Your command name, any parameters, and an optional Recordset.
cnnCompany.yourCommandName "parameter", rst
```

To execute a stored procedure, issue a statement where the stored procedure name is used as if it were a method on the Connection object, followed by any parameters. ADO will make a "best guess" of parameter types. For example:

```vba
Dim cnnCompany As New ADODB.Connection
...
'Your stored procedure name and any parameters.
cnnCompany.sp_yourStoredProcedureName "parameter"
```

**Recordset Object**

A Recordset object represents the entire set of records from a base table or the results of an executed command. At any time, the Recordset object refers to only a single record within the set as the current record.

**Remarks**

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

ADOR.Recordset and ADODB.Recordset are ProgIDs that you can use to create a Recordset object. The Recordset objects that result behave identically, regardless of the ProgID. The ADOR.Recordset is installed with Microsoft® Internet Explorer; the ADODB.Recordset is installed with ADO. The behavior of a Recordset object is affected by its environment (that is, client, server, Internet Explorer, and so on). Differences are noted in the Help topics for properties, methods, and events.

There are four different cursor types defined in ADO:

- **Dynamic cursor** — allows you to view additions, changes, and deletions by other users; allows all types of movement through the Recordset that doesn't rely on bookmarks; and allows bookmarks if the provider supports them.
- **Keyset cursor** — behaves like a dynamic cursor, except that it prevents you from seeing records that other users add, and prevents access to records that other users delete. Data changes by other users will still be visible. It always supports bookmarks and therefore allows all types of movement through the **Recordset**.

- **Static cursor** — provides a static copy of a set of records for you to use to find data or generate reports; always allows bookmarks and therefore allows all types of movement through the **Recordset**. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side (ADOR) **Recordset** object.

- **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 need to make only a single pass through a **Recordset**.

Set the **CursorType** property prior to opening the **Recordset** to choose the cursor type, or pass a **CursorType** argument with the **Open** method. Some providers don’t support all cursor types. Check the documentation for the provider. If you don't specify a cursor type, ADO opens a forward-only cursor by default.

If the **CursorLocation** property is set to **adUseClient** to open a **Recordset**, the **UnderlyingValue** property on **Field** objects is not available in the returned **Recordset** object. When used with some providers (such as the Microsoft ODBC Provider for OLE DB in conjunction with Microsoft SQL Server), you can create **Recordset** objects independently of a previously defined **Connection** object by passing a connection string with the **Open** method. ADO still creates a **Connection** object, but it doesn't assign that object to an object variable. However, if you are opening multiple **Recordset** objects over the same connection, you should explicitly create and open a **Connection** object; this assigns the **Connection** object to an object variable. If you do not use this object variable when opening your **Recordset** objects, ADO creates a new **Connection** object for each new **Recordset**, even if you pass the same connection string.

You can create as many **Recordset** objects as needed.

When you open a **Recordset**, the current record is positioned to the first record (if any) and the **BOF** and **EOF** properties are set to **False**. If there are no records, the **BOF** and **EOF** property settings are **True**.

You can use the **MoveFirst**, **MoveLast**, **MoveNext**, and **MovePrevious** methods, as well as the **Move** method, and the **AbsolutePosition**, **AbsolutePage**, and **Filter** properties, to reposition the current record, assuming the provider supports the relevant functionality. Forward-only **Recordset** objects support only the **MoveNext** method. When you use the **Move** methods to visit each record (or enumerate the **Recordset**), you can use the **BOF** and **EOF** properties to see if you've moved beyond the beginning or end of the **Recordset**.

**Recordset** objects can support two types of updating: immediate and batched. In immediate updating, all changes to data are written immediately to the underlying data source once you call the **Update** method. You can also pass arrays of values as parameters with the **AddNew** and **Update** methods and simultaneously update several fields in a record.

If a provider supports batch updating, you can have the provider cache changes to more than one record and then transmit them in a single call to the database with the **UpdateBatch** method. This applies to changes made with the **AddNew**, **Update**, and **Delete** methods. After you call the **UpdateBatch** method, you can use the **Status** property to check for any data conflicts in order to resolve them.

**Note** To execute a query without using a **Command** object, pass a query string to the **Open** method of a **Recordset** object. However, a **Command** object is required when you want to persist the command text and re-execute it, or use query parameters.
## ADO Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddNew</strong></td>
<td>Creates a new record for an updatable <strong>Recordset</strong> object.</td>
</tr>
<tr>
<td><strong>Append</strong></td>
<td>Appends an object to a collection. If the collection is <strong>Fields</strong>, a new <strong>Field</strong> object may be created before it is appended to the collection.</td>
</tr>
<tr>
<td><strong>AppendChunk</strong></td>
<td>Appends data to a large text or binary data <strong>Field</strong> or <strong>Parameter</strong> object.</td>
</tr>
</tbody>
</table>
| **BeginTrans**, **CommitTrans**, and **RollbackTrans** | Manages 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 and ends the current transaction. It may also start a new transaction. |
<p>| <strong>Cancel</strong>                  | Cancels execution of a pending, asynchronous <strong>Execute</strong> or <strong>Open</strong> method call. |
| <strong>Cancel (RDS)</strong>            | Cancels the currently running asynchronous execution or fetch.               |
| <strong>CancelBatch</strong>             | Cancels a pending batch update.                                             |
| <strong>CancelUpdate (ADO)</strong>      | Cancels any changes made to the current record or to a new record prior to calling the <strong>Update</strong> method. |
| <strong>CancelUpdate (RDS)</strong>      | Discards all the pending changes associated with the specified <strong>Recordset</strong> object, thus restoring the values since the last <strong>Refresh</strong> method call. |
| <strong>Clear</strong>                   | Removes all of the objects in a collection.                                |
| <strong>Clone</strong>                   | Creates a duplicate <strong>Recordset</strong> object from an existing <strong>Recordset</strong> object. Optionally, specifies that the clone be read-only. |
| <strong>Close</strong>                   | Closes an open object and any dependent objects.                            |
| <strong>CompareBookmarks</strong>        | Compares two bookmarks and returns an indication of their relative values.  |
| <strong>ConvertToString</strong>         | Converts a <strong>Recordset</strong> to a MIME string that represents the recordset data. |
| <strong>CopyRecord</strong>              | Copies a file or directory, and its contents, to another location.          |
| <strong>CopyTo</strong>                  | Copies the specified number of characters or bytes in the <strong>Stream</strong> to another <strong>Stream</strong> object. |
| <strong>CreateObject (RDS)</strong>      | Creates the proxy for the target business object and returns a pointer to it. |
| <strong>CreateParameter</strong>         | Creates a new <strong>Parameter</strong> object with the specified properties.           |
| <strong>CreateRecordset (RDS)</strong>   | Creates an empty, disconnected <strong>Recordset</strong>.                               |
| <strong>Delete (ADO Parameters Collection)</strong> | Deletes an object from the <strong>Parameters</strong> collection.                       |
| <strong>Delete (ADO Fields Collection)</strong> | Deletes an object from the <strong>Fields</strong> collection.                           |
| <strong>Delete (ADO Recordset)</strong>  | Deletes the current record or a group of records.                           |
| <strong>DeleteRecord</strong>            | Deletes a file or directory, and all its subdirectories.                    |
| <strong>Execute (ADO Command)</strong>   | Executes the query, SQL statement, or stored procedure specified in the <strong>CommandText</strong> property. |
| <strong>Execute (ADO Connection)</strong>| Executes the specified query, SQL statement, stored procedure, or provider-specific text. |
| <strong>Find</strong>                    | Searches a <strong>Recordset</strong> for the record that satisfies the specified criteria. |
| <strong>Flush</strong>                   | Forces the contents of the <strong>Stream</strong> remaining in the ADO buffer to the underlying object with which the <strong>Stream</strong> is associated. |
| <strong>GetChildren</strong>             | Returns a <strong>Recordset</strong> whose rows represent the files and subdirectories in the directory represented by this <strong>Record</strong>. |
| <strong>GetChunk</strong>                | Returns all or a portion of the contents of a large text or binary data <strong>Field</strong> object. |
| <strong>GetRows</strong>                 | Retrieves multiple records of a <strong>Recordset</strong> object into an array.         |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetString</td>
<td>Returns the Recordset as a string.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads the contents of an existing file into a Stream.</td>
</tr>
<tr>
<td>Move</td>
<td>Moves the position of the current record in a Recordset object.</td>
</tr>
<tr>
<td>MoveFirst, MoveLast, MoveNext, and MovePrevious (ADO)</td>
<td>Moves to the first, last, next, or previous record in a specified Recordset object and makes that record the current record.</td>
</tr>
<tr>
<td>MoveFirst, MoveLast, MoveNext, MovePrevious (RDS)</td>
<td>Moves to the first, last, next, or previous record in a displayed Recordset.</td>
</tr>
<tr>
<td>MoveRecord</td>
<td>Moves a file, or a directory and its contents, to another location.</td>
</tr>
<tr>
<td>NextRecordset</td>
<td>Clears the current Recordset object and returns the next Recordset by advancing through a series of commands.</td>
</tr>
<tr>
<td>Open (ADO Connection)</td>
<td>Opens a connection to a data source.</td>
</tr>
<tr>
<td>Open (ADO Record)</td>
<td>Opens an existing Record object, or creates a new file or directory.</td>
</tr>
<tr>
<td>Open (ADO Recordset)</td>
<td>Opens a cursor.</td>
</tr>
<tr>
<td>Open (ADO Stream)</td>
<td>Opens a Stream object to manipulate streams of binary or text data.</td>
</tr>
<tr>
<td>OpenSchema</td>
<td>Obtains database schema information from the provider.</td>
</tr>
<tr>
<td>Query (RDS)</td>
<td>Uses a valid SQL query string to return a Recordset.</td>
</tr>
<tr>
<td>Read</td>
<td>Reads a specified number of bytes from a Stream object.</td>
</tr>
<tr>
<td>ReadText</td>
<td>Reads a specified number of characters from a text Stream object.</td>
</tr>
<tr>
<td>Refresh (ADO)</td>
<td>Updates the objects in a collection to reflect objects available from and specific to the provider.</td>
</tr>
<tr>
<td>Refresh (RDS)</td>
<td>Requeries the ODBC data source specified in the Connect property and updates the query results.</td>
</tr>
<tr>
<td>Requery</td>
<td>Updates the data in a Recordset object by re-executing the query on which the object is based.</td>
</tr>
<tr>
<td>Reset (RDS)</td>
<td>Executes the sort or filter on a client-side Recordset based on the specified sort and filter properties.</td>
</tr>
<tr>
<td>Resync</td>
<td>Refreshes the data in the current Recordset object from the underlying database.</td>
</tr>
<tr>
<td>Save (ADO Recordset)</td>
<td>Saves (persists) the Recordset in a file or Stream object.</td>
</tr>
<tr>
<td>SaveToFile</td>
<td>Saves the binary contents of a Stream to a file.</td>
</tr>
<tr>
<td>Seek</td>
<td>Searches the index of a Recordset to quickly locate the row that matches the specified values, and changes the current row position to that row.</td>
</tr>
<tr>
<td>SetEOS</td>
<td>Sets the position that is the end of the stream.</td>
</tr>
<tr>
<td>SkipLine</td>
<td>Skips one entire line when reading a text Stream.</td>
</tr>
<tr>
<td>SubmitChanges (RDS)</td>
<td>Submits pending changes of the locally cached updatable Recordset to the ODBC data source specified in the Connect property.</td>
</tr>
<tr>
<td>Supports</td>
<td>Determines whether a specified Recordset object supports a particular type of functionality.</td>
</tr>
<tr>
<td>Update</td>
<td>Saves any changes you make to the current record of a Recordset object.</td>
</tr>
<tr>
<td>UpdateBatch</td>
<td>Writes all pending batch updates to disk.</td>
</tr>
<tr>
<td>Write</td>
<td>Writes binary data to a Stream object.</td>
</tr>
<tr>
<td>WriteText</td>
<td>Writes a specified text string to a Stream object.</td>
</tr>
</tbody>
</table>

**EXAMPLES**

**Ex1. Creating a Connection Object using CurrentProject Object**

Sub CreateConnection1()
    Dim cnnCompany As ADODB.Connection
    Set cnnCompany = New ADODB.Connection
    cnnCompany.Open CurrentProject.Connection
    'Your code goes here …
cnnCompany.Close
Set cnnCompany = Nothing
End Sub

---

**Ex2. Creating a Connection Object using a Connection String**

```vbscript
Sub CreateConnection2()
    Dim myConnectionString
    Dim cnnCompany As ADODB.Connection
    Set cnnCompany = New ADODB.Connection
    myConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;" & _
    "Data Source=C:\ADO-Notes-New\Company2K.mdb;" & _
    "Mode=Share Deny None;"
    cnnCompany.ConnectionString = myConnectionString
    cnnCompany.Open
    'Your code goes here ...
    cnnCompany.Close
    Set cnnCompany = Nothing
End Sub
```

---

**Ex3. Creating a Recordset using the CurrentProject Object**

```vbscript
Sub CreateRecordset1()
    Set rstCompany = New ADODB.Recordset
    rstCompany.Open "Select * From Employee", CurrentProject.Connection
    'quick and dirty displaying
    MsgBox rstCompany.GetString
    rstCompany.Close
    Set rstCompany = Nothing
End Sub
```

---

**Ex4. Creating a Recordset using the ActiveConnection Property**

```vbscript
Sub CreateRecordset2()
    Dim rstCompany As ADODB.Recordset
    Set rstCompany = New ADODB.Recordset
    rstCompany.ActiveConnection = CurrentProject.Connection
    rstCompany.Open "Select * From Employee"
    MsgBox rstCompany.GetString
    rstCompany.Close
    Set rstCompany = Nothing
End Sub
```

---

**Ex5. Creating Multiple Recordsets using a Connection Object**

```vbscript
Sub CreateRecordset3()
    Dim cnnCompany As ADODB.Connection
    Dim rstCompany1 As ADODB.Recordset
    Dim rstCompany2 As ADODB.Recordset
    Set cnnCompany = New ADODB.Connection
    Set rstCompany1 = New ADODB.Recordset
    Set rstCompany2 = New ADODB.Recordset
    cnnCompany.ConnectionString = CurrentProject.Connection
    cnnCompany.Open
```
rstCompany1.ActiveConnection = cnnCompany
rstCompany2.ActiveConnection = cnnCompany

rstCompany1.Open "Select * From Employee"
rstCompany2.Open "Select * From Project"

MsgBox rstCompany1.GetString
MsgBox rstCompany2.GetString

rstCompany1.Close
rstCompany2.Close
cnnCompany.Close

Set rstCompany1 = Nothing
Set rstCompany2 = Nothing
Set cnnCompany = Nothing

End Sub

Ex6. Creating a Dynamic Recordset using the ActiveConnection Property

Sub RecordsetMovements()
    Set rstCompany = New ADODB.Recordset
    rstCompany.ActiveConnection = CurrentProject.Connection
    'set the cursor to DYNAMIC (other options are adOpenStatic, adOpenKeySet,...)
    rstCompany.CursorType = adOpenStatic
    rstCompany.Open "Select * from Project"

    'note the mobility of the cursor
    rstCompany.moveFirst : MsgBox rstCompany("PName")
rstCompany.MoveNext : MsgBox rstCompany("PName")
rstCompany.MoveLast : MsgBox rstCompany("PName")
rstCompany.MovePrevious : MsgBox rstCompany("PName")
rstCompany.MoveFirst : MsgBox rstCompany("PName")

    rstCompany.Close
    Set rstCompany = Nothing
End Sub

Ex7A. Traversing a Static Recordset Sequentially

Sub DetermineLimits()
    Set rstCompany = New ADODB.Recordset
    rstCompany.ActiveConnection = CurrentProject.Connection

    'provides: forward motion + local update - other people's changes
    rstCompany.CursorType = adOpenForwardOnly

    rstCompany.Open "Select * from Employee where sex = 'F' "

    Do Until rstCompany.EOF
        MsgBox rstCompany("Lname")
rstCompany.MoveNext
Loop
rstCompany.MovePrevious  ‘ERROR: this invalid “move” should produce an exception
rstCompany.Close
Set rstCompany = Nothing
End Sub

Taken from VB Help File

CursorType Property

Indicates the type of cursor used in a Recordset object. Controls the motion (forward/backward) and the ability to see/block changes made by other users on the current data.

Sets or returns one of the following CursorTypeEnum values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adOpenForwardOnly</td>
<td>Forward-only cursor. Default. Identical to a static cursor except that you can only scroll forward through records. This improves performance in situations when you need to make only a single pass through a recordset.</td>
</tr>
<tr>
<td>adOpenKeyset</td>
<td>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.</td>
</tr>
<tr>
<td>adOpenDynamic</td>
<td>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.</td>
</tr>
<tr>
<td>adOpenStatic</td>
<td>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.</td>
</tr>
</tbody>
</table>

Remarks

Use the CursorType property to specify 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.

Only a setting of adUseStatic is supported if the CursorLocation property is set to adUseClient. If an unsupported value is set, then no error will result; the closest supported CursorType will be used instead.

If a provider does not support the requested cursor type, the provider may return another cursor type. The CursorType property will change to match the actual cursor type in use when the Recordset object is open. To verify specific functionality of the returned cursor, use the Supports method. After you close the Recordset, the CursorType property reverts to its original setting.

The following chart shows the provider functionality (identified by Supports method constants) required for each cursor type.

<table>
<thead>
<tr>
<th>For a Recordset of this CursorType</th>
<th>The Supports method must return True for all of these constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>adOpenForwardOnly</td>
<td>none</td>
</tr>
<tr>
<td>adOpenKeyset</td>
<td>adBookmark, adHoldRecords, adMovePrevious, adResync</td>
</tr>
<tr>
<td>adOpenDynamic</td>
<td>adMovePrevious</td>
</tr>
<tr>
<td>adOpenStatic</td>
<td>adBookmark, adHoldRecords, adMovePrevious, adResync</td>
</tr>
</tbody>
</table>

Note  Although Supports(adUpdateBatch) may be true for dynamic and forward-only cursors, for batch updates you should use either a keyset or static cursor. Set the LockType property to adLockBatchOptimistic and the CursorLocation property to adUseClient to enable the Cursor Service for OLE DB, which is required for batch updates.

Remote Data Service Usage  When used on a client-side (ADOR) Recordset object, the CursorType property can be set only to adOpenStatic.
Ex7B. Counting how many Records are held in a Recordset

Sub HowManyRecords()
    Dim rstCompany As ADODB.Recordset
    Set rstCompany = New ADODB.Recordset
    rstCompany.ActiveConnection = CurrentProject.Connection
    rstCompany.CursorType = adOpenStatic
    rstCompany.Open "Select * from Employee where ssn = 0 "
    MsgBox "absolute position: " & rstCompany.AbsolutePosition 'returns -1 when empty
    MsgBox "record count: " & rstCompany.RecordCount 'returns Recordcount
    rstCompany.Close
    Set rstCompany = Nothing
End Sub

Ex8. Sorting-filtering records on an existing Recordset

Sub SortFilterFindRecordset()
    Dim rstCompany As ADODB.Recordset
    Dim intCounter As Integer
    Dim thePerson, strCondition
    Set rstCompany = New ADODB.Recordset
    rstCompany.ActiveConnection = CurrentProject.Connection
    rstCompany.CursorLocation = adUseClient
    rstCompany.Open "Select * from Employee"

    'without filtering - Consider all employees
    rstCompany.Sort = "Salary ASC"
    MsgBox "Lowest Salary: " & rstCompany("Salary")
    rstCompany.Sort = "Salary DESC"
    MsgBox "Highest Salary: " & rstCompany("Salary")

    'filtering records - Consider only female employees
    rstCompany.Filter = "sex = 'F' "
    rstCompany.Sort = "Salary ASC"
    MsgBox "Female Lowest Salary: " & rstCompany("Salary")
    rstCompany.Sort = "Salary DESC"
    MsgBox "Female Highest Salary: " & rstCompany("Salary")

    'Find people whose name begins with W
    rstCompany.Filter = " Lname like 'W*' "
    If rstCompany.EOF Then
        MsgBox strCondition & " --- Nobody Found"
    Else
        While rstCompany.EOF = False
            MsgBox rstCompany("Lname") & " was Found "
            rstCompany.MoveNext
        Wend
    End If

    rstCompany.Close
    Set rstCompany = Nothing
End Sub
'Find people working for DNO = 5
rstCompany.Filter = " DNO = 5 "
If rstCompany.EOF Then
    MsgBox strCondition & " --- Nobody Found"
Else
    While rstCompany.EOF = False
        MsgBox "Works for DNO=5.  " & rstCompany("Lname")
        rstCompany.MoveNext
    Wend
End If

Ex9. Recordsets: Using BookMarks

Sub UseBookMark()
    'using BOOKMARKS to remember rec. positions
    Dim strCondition As String
    Dim vntPosition As Variant

    Set rstCompany = New ADODB.Recordset
    rstCompany.ActiveConnection = CurrentProject.Connection
    rstCompany.CursorType = adOpenStatic

    rstCompany.Open "Select * from Employee"

    'remember the position of the FIRST record
    vntPosition = rstCompany.Bookmark
    MsgBox "A bookmark is placed on: " & rstCompany("Lname")

    'show the Last Name of each employee
    Do Until rstCompany.EOF
        MsgBox rstCompany("Lname"), , "Visiting record"
        rstCompany.MoveNext
    Loop

    'return to the bookmarked position (First rec.)
    rstCompany.Bookmark = vntPosition
    MsgBox "Back to the bookmark: " & rstCompany("Lname")

    rstCompany.Close
    Set rstCompany = Nothing
End Sub

Ex10. Passing Parameters to an Executable Command Object

Sub ParameterizedCommand(theDept As Integer, theGender As String)
    'using ADO.Command ( test the sub using theDept = 4 & theGender = "F"
    Dim myLine, theDept, theGender

    Set cmd = New ADODB.Command
    cmd.ActiveConnection = CurrentProject.Connection

    'assemble the SQL query using its two parameters: theDEPT, and theGender
    cmd.CommandText = "select * from Employee where Dno = ? and sex = ? "
End Sub
cmd.CommandType = adCmdText
Set rstCompany = cmd.Execute(Parameters:=Array(theDept, theGender))

myLine = ""
Do Until rstCompany.EOF
    myLine = myLine & "Dno: " & rstCompany("Dno") & " " & rstCompany("Fname") & vbCrLf
    rstCompany.MoveNext
Loop
MsgBox myLine

rstCompany.Close
Set rstCompany = Nothing
Set cmd = Nothing
End Sub

Ex11. Updating a Recordset

Sub raiseLadiesSalary()
    'updating a recordset
    Set rstCompany = New ADODB.Recordset
    Dim strSQL As String
    Dim intUpdated As Integer

    rstCompany.ActiveConnection = CurrentProject.Connection
    rstCompany.CursorType = adOpenDynamic
    rstCompany.LockType = adLockOptimistic
    rstCompany.Open "Select ssn, fname, salary from Employee where Sex = 'F'  "

    MsgBox rstCompany.GetString, , "Old Salaries"
    rstCompany.MoveFirst

    'raise by 10% the salary of female employees
    While Not rstCompany.EOF
        intUpdated = intUpdated + 1
        rstCompany("Salary") = rstCompany("Salary") * 1.1
        rstCompany.Update
        rstCompany.MoveNext
    Wend

    MsgBox intUpdated & " Records Updated"
    rstCompany.Close
    rstCompany.Open "Select ssn, fname, salary from Employee where Sex = 'F'  "
    MsgBox rstCompany.GetString, , "New Salaries"
    rstCompany.Close
    Set rstCompany = Nothing
End Sub

Ex12. Deleting Rows from a Recordset

Sub DeleteFemaleEmployees()
    'deleting records using a recordset

    Dim myConn As ADODB.Connection
    Dim rstCompany As ADODB.Recordset
Dim intCounter As Integer

'define a connection to make possible a rollback (undelete recs.)
Set myConn = CurrentProject.Connection
'mark this point as the beginning of the transaction
myConn.BeginTrans

Set rstCompany = New ADODB.Recordset
rstCompany.ActiveConnection = myConn
rstCompany.CursorType = adOpenDynamic
rstCompany.LockType = adLockOptimistic

rstCompany.Open "Select * from Employee"

intCounter = 0
While Not rstCompany.EOF
    If rstCompany("Sex") = "F" Then
        MsgBox rstCompany("Fname"), , "To be deleted"
        rstCompany.Delete
        intCounter = intCounter + 1
    End If
    rstCompany.MoveNext
Wend

MsgBox intCounter & " Employee(s) Deleted"

rstCompany.Close
Set rstCompany = Nothing

'Rollback database to restore the deleted employees. The command
' myConn.CommitTrans makes permanent the removal of records
myConn.RollbackTrans
End Sub

Ex13. Inserting a new row into a Recordset

Sub newWorkLoad(theSSN As Long, theProject As Integer, theHours As Integer)
    Dim rstCompany As ADODB.Recordset
    Dim intCounter As Integer

    Set rstCompany = New ADODB.Recordset
    rstCompany.ActiveConnection = CurrentProject.Connection
    rstCompany.CursorType = adOpenStatic
    rstCompany.LockType = adLockOptimistic

    rstCompany.Open "Select * from Works_On"
    MsgBox rstCompany.RecordCount
    rstCompany.AddNew
        rstCompany("ESSN") = theSSN
        rstCompany("Pno") = theProject
        rstCompany("Hours") = theHours
    rstCompany.Update
    MsgBox rstCompany.RecordCount
    rstCompany.Close

    Set rstCompany = Nothing
End Sub
Ex14. Bulk-Update of a Recordset

Sub bulkUpdate()
    Dim cnnCompany As ADODB.Connection
    Dim mySQL, countRecAffected

    Set cnnCompany = New ADODB.Connection
    cnnCompany.Open CurrentProject.Connection

    mySQL = "Update Employee set Salary = Salary * 1.10 where sex = 'F' "
    cnnCompany.Execute mySQL, countRecAffected

    MsgBox countRecAffected & " records changed"
    cnnCompany.Close
    Set cnnCompany = Nothing
End Sub

Ex18. Running a Stored SQL-Query

Sub RunUpdateQuery()
    Dim cnnCompany As ADODB.Connection
    Set cnnCompany = New ADODB.Connection

    cnnCompany.Open CurrentProject.Connection
    cnnCompany.Execute "qryRaiseSalary"
    cnnCompany.Close
End Sub

Ex19. Exploring the Database Structure Using ADOX

Sub exploringADOX()
    '--- To include ADOX do as follows
    '--- Tools | References | Choose “Microsoft ADO Ext. 2.5 for DDL and Security"

    Dim cat As ADOX.Catalog
    Set cat = New ADOX.Catalog
    Dim msg

    cat.ActiveConnection = CurrentProject.Connection
    For Each t In cat.Tables
        msg = t.Name & vbTab & t.Type & vbCrLf
        For Each f In t.Columns
            msg = msg & f.Type & vbTab & f.Name & vbCrLf
        Next
        MsgBox msg
    Next
End Sub
Using DAO Control

DAO Overview
DAO enables you to use a programming language to access and manipulate data in local or remote databases, and to manage databases, their objects, and their structure.

Object Models
DAO supports two different database environments, or "workspaces."

- **Microsoft Jet workspaces** allow you to access data in Microsoft Jet databases, Microsoft Jet-connected ODBC data sources, and installable ISAM data sources in other formats, such as Paradox or Lotus 1-2-3.

- **ODBCDirect workspaces** allow you to access database servers through ODBC, without loading the Microsoft Jet database engine.

Use the Microsoft Jet workspace when you open a Microsoft Jet database (.mdb file) or other desktop ISAM database, or when you need to take advantage of Microsoft Jet's unique features, such as the ability to join data from different database formats.

The ODBCDirect workspace provides an alternative when you only need to execute queries or stored procedures against a back-end server, such as Microsoft SQL Server, or when your client application needs the specific capabilities of ODBC, such as batch updates or asynchronous query execution.

DAO Objects
There are 17 different DAO object types. You can declare new DAO **object variables** for any of the object types.

For example, the following Visual Basic for Applications (VBA) code creates object variables for a **Database** object, a dynaset-type **Recordset** object, and a **Field** object:

```vba
Dim dbsExample As Database
Dim rstExample As Recordset
Dim fldExample As Field

Set dbsExample = OpenDatabase("Biblio.mdb")
Set rstExample = dbsExample.OpenRecordset("Authors", dbOpenDynaset)
Set fldExample = rstExample.Fields("Au_ID")
```

DAO Collections
Each DAO object type other than **DBEngine** also has a corresponding collection. A collection includes all the existing objects of that type. For example, the **Recordsets** collection contains all open **Recordset** objects. Each collection is "owned" by another object at the next higher level in the hierarchy. A **Database** object "owns" a **Recordsets** collection. Except for the **Connection** and **Error** objects, every DAO object has a **Properties** collection.

Most DAO objects have default collections and default properties. For example, the default collection of a **Recordset** object is the **Fields** collection and the default property of a **Field** object is the **Value** property. You can simplify your code by taking advantage of these defaults. For example, the following code sets the value of the PubID field in the current record:

```vba
rstExample!PubID=99  (you can use the equivalent syntax  rstExample("PubID") = 99 )
```
DBENGINE AND WORKSPACE OBJECTS

All DAO objects are derived from the DBEngine object. You can set the DefaultType property on the DBEngine object to determine the workspace type (Microsoft Jet or ODBCDirect) to create on subsequent CreateWorkspace method calls, or you can override this property with the type argument in the CreateWorkspace method itself. When your application creates a workspace, the appropriate library — the Microsoft Jet database engine or ODBC — is loaded into memory at that time.

You can open additional Workspace objects as needed. Each Workspace object has a user ID and password associated with it.

Using the Microsoft Jet Workspace

Opening a Database

To open a database, you simply open an existing Database object, or create a new one. This object can represent a Microsoft Jet database (.mdb file), an ISAM database (for example, Paradox), or an ODBC database connected through the Microsoft Jet database engine (also known as a "Microsoft Jet-connected ODBC database").

Data-Definition Language

You can use object variables and other DDL features to modify your database structure. For example, you can add a new Field object to an existing table with the following code:

```
Dim dbs As Database, tdf As TableDef, fld As Field
' Open a database.
Set dbs = OpenDatabase("Biblio.mdb")
' Open a TableDef.
Set tdf = dbs.TableDefs("Authors")
' Create a new field.
Set fld = tdf.CreateField("Address", dbText, 20)
' Append field to the TableDef Fields collection.
```

tdf.Fields.Append fld

This code creates a new object variable for a Field object and adds it to a TableDef object with the Append method. Because a TableDef object contains the definition of a table, the table now has a field named Address for entering data. In much the same way, you can create new tables and new indexes.

Data Manipulation

DAO provides an excellent set of data manipulation tools. You can create a Recordset object to conveniently query a database and manipulate the resulting set of records. The OpenRecordset method accepts an SQL string, or a QueryDef (stored query) name as a data source argument, or it can be opened from a QueryDef object or a TableDef object, using that object as its data source. The resulting Recordset object features an extremely rich set of properties and methods with which to browse and modify data.

The Recordset object is available in four different types — Table, Dynaset, Forward-Only, and Snapshot.

Transactions

All Database objects opened against a Workspace object share a common transaction scope. That is, when you use the BeginTrans method on a Workspace object, it applies to all open databases within that Workspace object. In the same way, when you use the CommitTrans method against the Workspace, it applies to all open databases in the Workspace object.

Replication

You can use database replication to create and maintain replicas of a master Microsoft Jet database, using the Synchronize method to periodically update all or part of the replicas, or to copy new data from one replica to another. You can also restrict the update to only selected records, using the ReplicaFilter property, and then synchronize those records with the PopulatePartial method.
Security

You can restrict access to one or more .mdb databases or their tables using security settings established and managed by the Microsoft Jet database engine. In your code, you can establish Group and User objects to define the scope and level of permissions available to individual users on an object-by-object basis. For example, you can establish permissions for a specific user to provide read-only access to one table and full access to another.

Using the ODBCDirect Object Model

Connecting to a Database

A Connection object is similar to a Database object. In fact, a Connection object and a Database object represent different references to the same object, and properties on each of these two object types allow you to obtain a reference to the other corresponding object, which simplifies the task of converting ODBC client applications that use Microsoft Jet to use ODBCDirect instead. Use the OpenConnection method to connect to an ODBC data source. The resulting Connection object contains information about the connection, such as the server name, the data source name, and so on.

Queries

Although DAO does not support stored queries in an ODBCDirect workspace, a compiled query can be created as a QueryDef object and used to execute action queries, and can also be used to execute stored procedures on the server. The Prepare property lets you decide whether to create a private, temporary stored procedure on the server from a QueryDef before actually executing the query.

Parameter queries can also be passed to the server, using Parameter objects on the QueryDef. The Direction property lets you specify a Parameter as input, output, or both, or to accept a return value from a stored procedure.

Data Manipulation

Creating a Recordset object is a convenient way to query a database and manipulate the resulting set of records. The OpenRecordset method accepts an SQL string, or a QueryDef object (stored query) as a data source argument. The resulting Recordset object features an extremely rich set of properties and methods with which to browse and modify data.

The Recordset object is available in four different types — Dynamic, Dynaset, Forward-Only, and Snapshot — corresponding to ODBC cursor types — Dynamic, Keyset, Forward-only, and Static.

A batch update cursor library is available for client applications that need to work with a cursor without holding locks on the server or without issuing update requests one record at a time. Instead, the client stores update information on many records in a local buffer (or "batch"), and then issues a batch update.

Asynchronous Method Execution

The Execute, MoveLast, OpenConnection, and OpenRecordset methods feature the dbRunAsync option. This allows your client application to do other tasks (such as loading forms, for example) while the method is executing. You can check the StillExecuting property to see whether the task is complete, and terminate an asynchronous task with the Cancel method.
DAO Objects and Collections Reference

DAO objects and collections provide a framework for using code to create and manipulate components of your database system. Objects and collections have properties that describe the characteristics of database components and methods that you use to manipulate them. Together these objects and collections form a hierarchical model of your database structure, which you can control programmatically.

Objects and collections provide different types of containment relations: Objects contain zero or more collections, all of different types; and collections contain zero or more objects, all of the same type. Although objects and collections are similar entities, the distinction differentiates the two types of relations.

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.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>Connection</td>
<td>Information about a connection to an ODBC data source (ODBCDirect workspaces only)</td>
</tr>
<tr>
<td>Containers</td>
<td>Container</td>
<td>Storage for information about a predefined object type (Microsoft Jet workspaces only)</td>
</tr>
<tr>
<td>Databases</td>
<td>Database</td>
<td>An open database</td>
</tr>
<tr>
<td>None</td>
<td>DBEngine</td>
<td>The Microsoft Jet database engine</td>
</tr>
<tr>
<td>Documents</td>
<td>Document</td>
<td>Information about a saved, predefined object (Microsoft Jet workspaces only)</td>
</tr>
<tr>
<td>Errors</td>
<td>Error</td>
<td>Information about any errors associated with this object</td>
</tr>
<tr>
<td>Fields</td>
<td>Field</td>
<td>A column that is part of a table, query, index, relation, or recordset</td>
</tr>
<tr>
<td>Groups</td>
<td>Group</td>
<td>A group of user accounts (Microsoft Jet workspaces only)</td>
</tr>
<tr>
<td>Indexes</td>
<td>Index</td>
<td>Predefined ordering and uniqueness of values in a table (Microsoft Jet workspaces only)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Parameter</td>
<td>A parameter for a parameter query</td>
</tr>
<tr>
<td>Properties</td>
<td>Property</td>
<td>A built-in or user-defined property</td>
</tr>
<tr>
<td>QueryDefs</td>
<td>QueryDef</td>
<td>A saved query definition</td>
</tr>
<tr>
<td>Recordsets</td>
<td>Recordset</td>
<td>The records in a base table or query</td>
</tr>
<tr>
<td>Relations</td>
<td>Relation</td>
<td>A relationship between fields in tables and queries (Microsoft Jet workspaces only)</td>
</tr>
<tr>
<td>TableDefs</td>
<td>TableDef</td>
<td>A saved table definition (Microsoft Jet workspaces only)</td>
</tr>
<tr>
<td>Users</td>
<td>User</td>
<td>A user account (Microsoft Jet workspaces only)</td>
</tr>
<tr>
<td>Workspaces</td>
<td>Workspace</td>
<td>A session of the Microsoft Jet database engine</td>
</tr>
</tbody>
</table>
Using DAO to Explore the architecture of a database

Public Sub ExploringDBArchitecture()
On Error Resume Next

    Dim db As Database
    Dim tbl As DAO.TableDef
    Dim idx As DAO.Index
    Dim fld As DAO.Field
    Dim qry As QueryDef
    Dim prm As Parameter
    Dim rel As Relation
    Dim cnt As Container
    Dim doc As Document

    '---Get access to the current database
    Set db = CurrentDb

    '--- Show TABLES-FIELDS and INDEXES
    For Each tbl In db.TableDefs
        If Mid(tbl.Name, 1, 4) <> "MSys" Then
            Debug.Print "Table: "; tbl.Name
            '--- Show FIELDS of current table
            For Each fld In tbl.Fields
                Debug.Print fld.Name & " " & fld.Type
            Next fld
            '---Show INDEXES
            For Each idx In tbl.Indexes
                Debug.Print "  Index: "; idx.Name
                For Each fld In idx.Fields
                    Debug.Print "  Index Field:"; fld.Name
                Next fld
            Next idx
        End If
    Next tbl

    '--- List QUERY collection
    For Each qry In db.QueryDefs
        Debug.Print "Query: "; & qry.Name
        Debug.Print qry.SQL
        '--- Show PARAMETERS
        For Each prm In qry.Parameters
            Debug.Print "  Parameter: "; & prm.Name
        Next prm
    Next qry

    '--- Show RELATIONSHIPS
    For Each rel In db.Relations
        Debug.Print rel.Table & " Related To: "; & rel.ForeignTable
    Next rel

    '--- Show CONTAINERS
    For Each cnt In db.Containers
        Debug.Print cnt.Name
    Next cnt

    '--- Show FORMS
    Set cnt = db.Containers!Forms
    For Each doc In cnt/Documents
Using the DAO control

Sub OpenTable()
    '--- Give a $1000 bonus to female employees
    Dim daoDB As DAO.Database
    Dim daoRS As DAO.Recordset
    Dim mySQL, msg, theGender

    Set daoDB = CurrentDb()
    theGender = "F"
    mySQL = "select * from Employee where sex = " & theGender & ""
    Set daoRS = daoDB.OpenRecordset(mySQL)
    msg = ""
    While Not daoRS.EOF
        msg = msg & daoRS("Fname") & vbTab & daoRS("Salary") & vbTab
        daoRS.Edit
        daoRS("Salary") = daoRS("Salary") + 1000
        daoRS.Update
        msg = msg & daoRS("Salary") & vbCrLf
        daoRS.MoveNext
    Wend
    MsgBox msg
End Sub