

# CHAPTER 30

## UNDERSTANDING UNIVERSAL DATA ACCESS, OLE DB, AND ADO

### In this chapter

- Gaining a Perspective on Microsoft Data Access Components 1258
- Creating `ADODB.Recordset` Objects 1265
- Using the Object Browser to Display ADO Properties, Methods, and Events 1271
- Working with the `ADODB.Connection` Object 1273
- Using the `ADODB.Command` Object 1286
- Understanding the `ADODB.Recordset` Object 1296
- Taking Advantage of Disconnected Recordsets 1312
- Programming Stream Objects 1323
- Exploring the `AddOrder.adp` Sample Project 1327
- Troubleshooting 1330
- In the Real World—Why Learn ADO Programming? 1330

## GAINING A PERSPECTIVE ON MICROSOFT DATA ACCESS COMPONENTS

Integrated data management is the key to Access's success in the desktop RDBMS and client/server front-end market. Access and its wizards let you create basic data-bound forms, reports, and pages with minimal effort and little or no VBA programming. Linked tables provide dynamic access to a wide range of data sources. As your Access applications grow larger and more complex, automation with VBA code in class and public modules becomes essential. When networked Access applications gain more users, performance may suffer as a result of Jet record-locking issues or multiple connections to client/server back ends.

Decreasing performance with increasing user load is a symptom of lack of *scalability*. Achieving scalability requires VBA code to manage your application's database connections. This advanced chapter shows you how to write the VBA code that's required to improve the scalability of Access front ends. You also learn how to use the `Stream` object to generate XML data documents from SQL Server 2000's `FOR XML AUTO` queries.

Access 2003 continues Microsoft's emphasis on "Universal Data Access" for VBA and Visual Basic 6.0 programmers. Microsoft wants Access developers to abandon Jet's Data Access Objects (DAO), Access 97's ODBCDirect, and the venerable Open Database Connectivity (ODBC) Application Programming Interface (API) in favor of a collection of Component Object Model (COM) interfaces called OLE DB and ActiveX Data Objects (ADO). To encourage Access power users and developers to adopt OLE DB and ADO, all traditional Microsoft database technologies (referred to by Microsoft as *downlevel* or *legacy*, synonyms for "obsolete") are destined for maintenance mode. Maintenance mode is a technological purgatory in which Microsoft fixes only the worst bugs and upgrades occur infrequently, if ever. In 1999, OLE DB, ADO, and, for Jet programmers, ActiveX Data Object Extensions (ADOX), became Microsoft's mainstream data access technologies.

Microsoft's primary goals for Universal Data Access were to

- Provide the capability to accommodate less common data types unsuited to SQL queries, such as directory services (specifically Active Directory), spreadsheets, email messages, and file systems
- Minimize the size and memory consumption of the dynamic link libraries (DLLs) required to support data access on Internet and intranet clients
- Reduce development and support costs for the multiplicity of Windows-based data access architectures in common use today
- Extend the influence of COM in competition with other object models, primarily Common Object Request Broker Architecture (CORBA) and its derivatives

This chapter introduces you to the fundamentals of Universal Data Access and Microsoft Data Access Components (MDAC). MDAC makes connecting to databases with OLE DB practical for Access users and developers. MDAC includes ADO and ADOX for conventional relational data, plus ADOMD for multidimensional expressions (MDX) to create and manipulate data cubes.

**NOTE**

Microsoft SQL Server Analysis Services (formerly OLAP Services) generates data cubes from online sources, such as transactional databases. Office 2003 installs Msadomd.dll and other supporting files for MDX and data cubes. Microsoft provides OLE DB for OLAP and the PivotTable Service to enable Excel 2003 PivotTables to manipulate data cubes. MDX and PivotTable services are beyond the scope of this book.

**REDESIGNING FROM THE BOTTOM UP WITH OLE DB**

To accommodate the widest variety of data sources, as well as to spread the gospel of COM and Windows XP/2000+'s COM+, Microsoft's data architects came up with a new approach to data connectivity—OLE DB. OLE DB consists of three basic elements:

- *Data providers* that abstract information contained in data sources into a tabular (row-column) format called a *rowset*. Microsoft currently offers native OLE DB data providers for Jet, SQL Server, IBM DB2, IBM AS/400 and ISAM, and Oracle databases, plus ODBC data sources. (Only Microsoft SNA Server installs the providers for IBM data sources.) Other Microsoft OLE DB providers include an OLE DB Simple Provider for delimited text files, the MSPersist provider for saving and opening Recordsets to files (called *persisted Recordsets*), and the MSDataShape provider for creating hierarchical data sets. The MSDataShape provider also plays an important role in ADP and when using VBA to manipulate the Recordset of Access forms and reports.

**TIP**

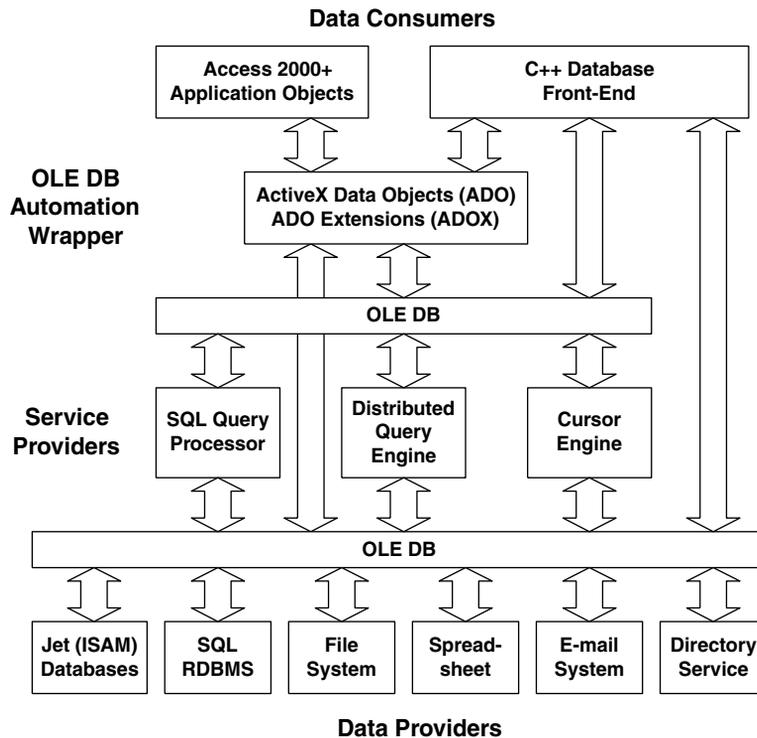
To see the list of OLE DB data sources installed on your computer, open the NorthwindCS.adp project, and choose File, Get External Data, Link Tables to start the Link Table Wizard. With the Linked Server option selected in the first dialog, click Next to open the Select Data Source dialog, and double-click the +Connect to New Data Source.odc file to open the second Wizard dialog. With the Other/Advanced item selected in the data source list, click Next to open the Data Link Properties dialog. The Providers page lists all currently installed OLE DB data providers. Click Cancel three times to return to the Database window.

- *Data consumers* that display and/or manipulate rowsets, such as Access application objects or OLE DB service providers. Rowset is the OLE DB object that ADO converts to a Recordset object.
- *Data services* (usually called *OLE DB service providers*) that consume data from providers and, in turn, provide data to consumers. Examples of data services are SQL query processors and cursor engines, which can create scrollable rowsets from forward-only rowsets. A scrollable cursor lets you move the record pointer forward and backward in the Datasheet view of a Jet or SQL Server query.

Figure 30.1 illustrates the relationship between OLE DB data providers, data consumers, and data services within Microsoft's Universal Data Access architecture. You should understand the relationships between these objects, because Microsoft commonly refers to them in ADO documentation, help files, and Knowledge Base articles. Database front ends written in C++ can connect directly to the OLE DB interfaces. High-level languages, such as VBA, use ADO as an intermediary to connect to OLE DB's COM interfaces. Msado15.dll, which implements ADO 2.x, has a memory footprint of about 327KB, about 60% of Dao360.dll's 547KB.

**Figure 30.1**

This diagram shows the relationships between front-end applications, ADO and ADOX, and OLE DB service and data providers.



ADO support files install in your \Program Files\System\Ado folder. If you're running Windows XP/2000+, the ADO support files are subject to Windows File Protection (WFP), which places a copy of the file in the DLL cache and prevents you from permanently deleting or moving the ADO support files. WFP also prevents unruly installation programs from overwriting the ADO support files with an earlier or corrupt (hacked) version.

Some ADO 2.x support file names have a 1.5 version number, as in Msado15.dll; the strange versioning of these files is required for backward compatibility with applications that used very early versions of ADO.

**NOTE**

MDAC 2.x also supports Remote Data Services (RDS, formerly Advanced Database Connector, or ADC). RDS handles lightweight ADOR.Recordsets for browser-based applications; RDS, which commonly is used for three-tier, Web-based applications, is required to make Data Access Pages (DAP) accessible safely over the Internet.

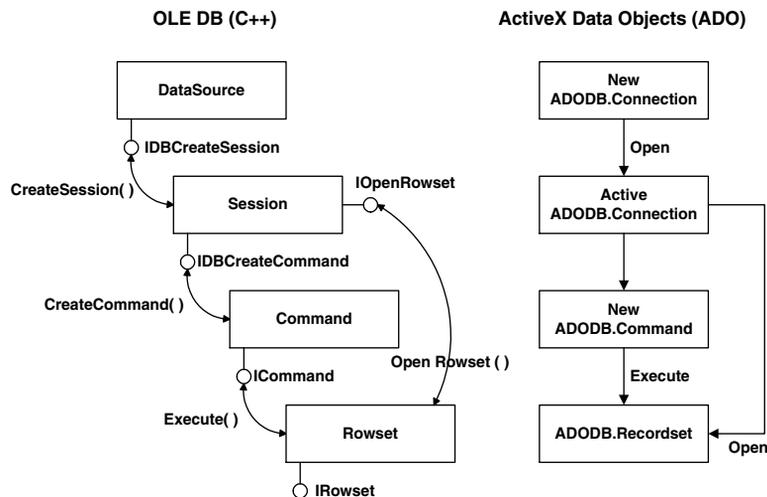
→ For more information on the use of RDS with DAP, see “Enabling Private or Public Internet Access,” p. 1058.

**MAPPING OLE DB INTERFACES TO ADO**

You need to know the names and relationships of OLE DB interfaces to ADO objects, because Microsoft includes references to these interfaces in its technical and white papers on OLE DB and ADO. Figure 30.2 illustrates the correspondence between OLE DB interfaces and the highest levels of the ADO hierarchy.

**Figure 30.2**

This diagram illustrates the correspondence between OLE DB interfaces and ADO Automation objects.



The OLE DB specification defines a set of interfaces to the following objects:

- DataSource objects provide a set of functions to identify a particular OLE DB data provider, such as the Jet or SQL Server provider, and determine whether the caller has the required security permissions for the provider. If the provider is found and authentication succeeds, a connection to the data source results.
- Session objects provide an environment for creating rowsets and isolating transactions, especially with Microsoft Transaction Server (MTS), which runs under Windows NT. The COM+ components of Windows 2000+ provide MTS services.
- Command objects include sets of functions to handle queries, usually (but not necessarily) in the form of SQL statements or names of stored procedures.

- Rowset objects can be created directly from Session objects or as the result of execution of Command objects. Rowset objects deliver data to the consumer through the IRowset interface.

ADO maps the four OLE DB objects to the following three top-level Automation objects that are familiar to Access programmers who've used ODBCDirect:

- Connection objects combine OLE DB's DataSource and Session objects to specify the OLE DB data provider, establish a connection to the data source, and isolate transactions to a specific connection. The Execute method of the ADODB.Connection object can return a forward-only ADODB.Recordset object.
- Command objects are directly analogous to OLE DB's Command object. ADODB.Command objects accept an SQL statement, the name of a table, or the name of a stored procedure. Command objects are used primarily for executing SQL UPDATE, INSERT, DELETE, and SQL Data Definition Language (DDL) queries that don't return records. You also can return an ADODB.Recordset by executing an ADODB.Command object.
- Recordset objects correspond to OLE DB's Rowset objects and have properties and methods similar to Access 97's ODBCDirect Recordset. A Recordset is an in-memory image of a table or a query result set.

The ADODB prefix, the short name of the ADO type library, explicitly identifies ADO objects that share object names with DAO (Recordset) and DAO's ODBCDirect (Connection and Recordset). For clarity, all ADO code examples in this book use the ADODB prefix.

#### TIP

To make ADOX 2.7 accessible to VBA, you must add a reference to Microsoft ADO Ext. 2.7 for DDL and Security to your application. Access 2003 doesn't add the ADOX reference automatically to new projects.

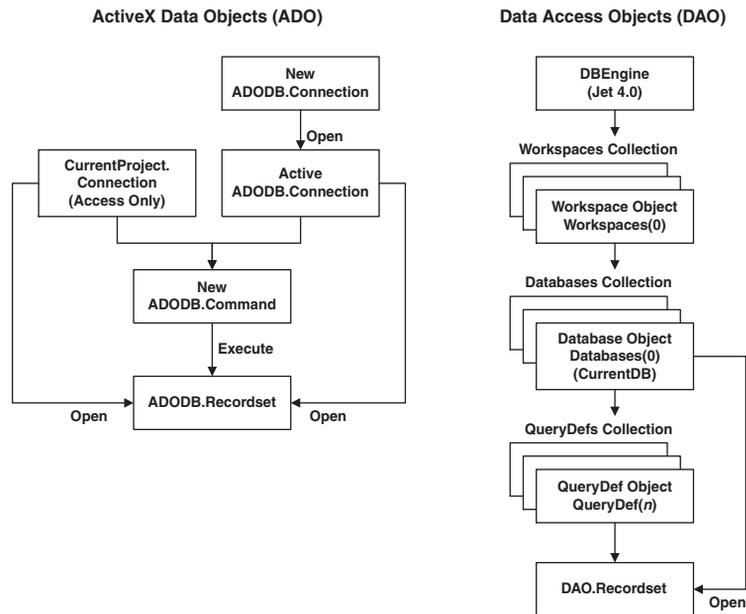
## COMPARING ADO AND DAO OBJECTS

Figure 30.3 is a diagram that compares the ADO and DAO object hierarchies. The ADO object hierarchy, which can consist of nothing more than an ADODB.Connection object, is much simpler than the collection-based object hierarchy of DAO. To obtain a scrollable, updatable Recordset (dynaset), you must open an ADODB.Recordset object on an active ADODB.Connection object.

Access VBA provides a DAO shortcut, `Set dbName = CurrentDB()`, to bypass the first two collection layers and open the current database, but `CurrentDB()` isn't available in VBA code for other members of Office 2003 or Visual Basic 6.0.

**Figure 30.3**

This diagram compares the ADO and DAO object hierarchies.

**NOTE**

Access VBA provides a similar ADO shortcut, `CurrentProject.Connection`, which points to a default `ADODB.Connection` object with the Jet OLE DB Service Provider for the current database. Unlike `CurrentDB()`, which is optional, you must use `CurrentProject.Connection` as the `ADODB.Connection` to the currently open database. If you try to open a new `ADODB.Connection` to the current database, you receive a runtime error stating that the database is locked.

Unlike DAO objects, most of which are members of collections, you use the **New** reserved word with the **Set** instruction to create and the `Close` method, the **Set** `ObjectName = Nothing`, or both statements to remove instances of `ADODB.Connection`, `ADODB.Command`, and `ADODB.Recordset` objects independently of one another. The **Set** `ObjectName = Nothing` instruction releases memory consumed by the object.

DAO supports a variety of Jet collections, such as `Users` and `Groups`, and Jet SQL Data Definition Language (DDL) operations that ADO 2.7 alone doesn't handle. ADOX 2.7 defines Jet-specific collections and objects that aren't included in ADO 2.x. The "Provider-Specific Properties and Their Values" section later in the chapter describes how to roll your own cross-reference table to aid in migrating your DAO code to ADO.

The most important functional difference between DAO and ADO is that ADO supports Web-based applications and DAO doesn't. Thus, DAP bind to `ADODB.Recordset` objects. The continuing trend toward Internet-enabling everything means that Windows database

programmers must make the transition from ODBC, ODBCDirect, RDO, and DAO technologies to ADO and OLE DB, so this book covers VBA programming of ADO, not DAO, objects. ADO supports ODBC connections to shared-file and client/server RDBMSs with the Microsoft OLE DB Provider for ODBC (more commonly called by its beta code name, *Kagera*). ODBC introduces another layer into the database connection, so it's less efficient than OLE DB. The examples of this chapter use only native OLE DB providers.

## UPGRADING FROM ADO 2.5 AND EARLIER TO VERSION 2.6+

*ADO 2.x* in this chapter refers collectively to ADO 2.1, 2.5, 2.6, and 2.7. Windows XP and Office 2003 install ADO 2.7, which includes type libraries for ADO 2.0, 2.1, 2.5, 2.6 for backward compatibility. Windows 2000 Service Pack (SP) 1 or later installs ADO 2.5 SP1, which includes type libraries for for prior versions. Installing the SQL Server 2000 Desktop Engine (MSDE2000) from the Office 2003 distribution CD-ROM—or any other version of SQL Server 2000—upgrades Windows 2000's ADO 2.5 to 2.6. Version 2.7 is required only to support Intel's 64-bit Itanium processors. Upgrading from ADO 2.6 to 2.7 doesn't add new features or alter existing features.

### NOTE



As mentioned in Chapter 27, "Learning Visual Basic for Applications," the default VBA reference for new ADP is ADO 2.1 for Access 2000 database format. If you change the default database version to Access 2002 in the Options dialog, the reference changes to ADO 2.5. Use of non-current references is required for backward compatibility with Access 2000 and 2002 ADP.

→ To review use of the VBA editor's References dialog, see "References to VBA and Access Modules," p. 1157.



Following are the new or altered ADO objects, properties, and methods in ADO 2.6+:

- Record objects can contain fields defined as `Recordsets`, Streams of binary or text data, and child records of hierarchical `Recordset` objects. Use of `Record` objects is beyond the scope of this book.
- Stream objects can send T-SQL FOR XML queries to SQL Server 2000 and return result sets as XML documents. Stream objects also are used with the `Record` object to return binary data from URL queries executed on file systems, Exchange 2000 Web Folders, and email messages. The "Programming Stream Objects" section, near the end of the chapter, provides a simple example of the use of a `Stream` object to return XML data from a FOR XML T-SQL query to a text box.

- Command objects gain new `CommandStream` and `Dialect` properties to support Stream objects, and a `NamedParameters` property that applies to the `Parameters` collection.
- Group and User ADOX objects add a `Properties` collection that contains Jet-specific Property objects. This chapter doesn't cover ADOX programming with VBA, because ADOX applies only to Jet databases.

**TIP**

If you're interested in learning more about ADOX, open the VBA Editor, type **adox** in the Ask a Question text box, select the ADOX methods option, click See Also in the "ADOX Methods" help page, and select ADOX API Reference in the list.

## CREATING ADODB.Recordset OBJECTS

The concept of database object independence is new to Access. The best way of demonstrating this feature is to compare DAO and ADO code to create a `Recordset` object from an SQL statement. DAO syntax uses successive instantiation of each object in the DAO hierarchy: `DBEngine`, `Workspace`, `Database`, and `Recordset`, as in the following example:

```
Dim wsName As DAO.Workspace
Dim dbName As DAO.Database
Dim rstName As DAO.Recordset

Set wsName = DBEngine.Workspaces(0)
Set dbName = wsName.OpenDatabase ("DatabaseName.mdb")
Set rstName = dbName.OpenRecordset ("SQL Statement")
```

As you descend through the hierarchy, you open new child objects with methods of the parent object.

The most common approach with ADO is to create one or more independent, reusable instances of each object in the Declarations section of a form or module:

```
Private cnnName As New ADODB.Connection
Private cmmName As New ADODB.Command
Private rstName As New ADODB.Recordset
```

**NOTE**

This book uses `cnn` as the object type prefix for `Connection`, `cmm` for `Command`, and `rst` for `Recordset`. The `cmm` prefix is used because the `cmd` prefix traditionally identifies a command button control and the `com` prefix identifies the `MSComm` ActiveX control (Microsoft Comm Control 6.0).

Although you're likely to find references to `DAO.Recordset` dynasets and snapshots in the Access documentation, these terms don't apply to `ADODB.Recordset` objects. See the `CursorType` property of the `ADODB.Recordset` object in the "Recordset Properties" section later in this chapter for the `CursorType` equivalents of dynasets and snapshots.

After the initial declarations, you set the properties of the new object instances and apply methods—Open for Connections and Recordsets, or Execute for Commands—to activate the object. Invoking the Open method of the ADODB.Recordset object, rather than the OpenRecordset method of the DAO.Database object, makes ADO objects independent of one another. Object independence and batch-optimistic locking, for instance, let you close the ADODB.Recordset's ADODB.Connection object, make changes to the Recordset, and then reopen the Connection to send only the changes to the underlying tables. Minimizing the number of open database connections conserves valuable server resources. The examples that follow illustrate the independence of top-level ADO members.

## DESIGNING A FORM BOUND TO AN ADODB.Recordset OBJECT

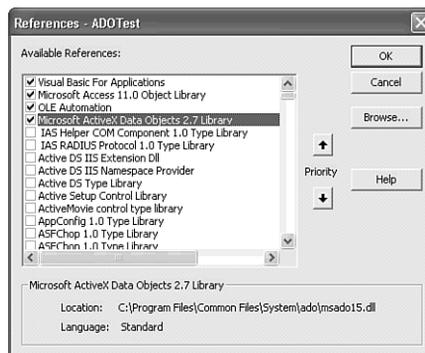
**NEW 2002** Access 2000+ forms have a property, Recordset, which lets you assign an ADODB.Recordset object as the RecordSource for one or more forms. The Recordset property of a form is an important addition, because you can assign the same Recordset to multiple forms. All forms connected to the Recordset synchronize to the same current record. Access developers have been requesting this feature since version 1.0. Access 2002 delivered updatable ADODB.Recordsets for Jet, SQL Server, and Oracle data sources that you can assign to the Recordset property value of forms and reports.

To create a simple form that uses VBA code to bind a form to a Jet ADODB.Recordset object, follow these steps:

-  **1.** Open a new database in Access 2000 format named **ADOTest.mdb** or the like in your ... \Office11 \Samples folder. Add a new form in design mode and save it as **frmADO\_Jet**.
-  **2.** Click the Code button on the toolbar to open the VBA editor, and choose **T**ools, **R**eferences to open the References dialog.
- NEW 2002** **3.** Clear the check box for the reference to the Microsoft ActiveX Data Objects 2.1 Library, scroll to the Microsoft ActiveX Data Objects 2.7 Library, and mark the check box. Close and reopen the References dialog to verify that the new reference has percolated to the select region of the list (see Figure 30.4). Close the References dialog.

**Figure 30.4**

If you don't need backward compatibility with Access 2000 and 2002 applications, specify the latest version of ADO (2.7 for this example) as the reference.



4. Add the following code to the Declarations section of the frmADO\_Jet Class Module:

```
Private strSQL As String
Private cnnNwind As New ADO DB .Connection
Private rstNwind As New ADO DB .Recordset
```

5. Add the following code to create the Form\_Load event handler:

```
Private Sub Form_Load()
    'Specify the OLE DB provider and open the connection
    With cnnNwind
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open CurrentProject.Path & "\Northwind.mdb", "Admin"
    End With

    strSQL = "SELECT * FROM Customers"
    With rstNwind
        Set .ActiveConnection = cnnNwind
        .CursorType = adOpenKeyset
        .CursorLocation = adUseClient
        .LockType = adLockOptimistic
        .Open strSQL
    End With

    'Assign rstNwind as the Recordset for the form
    Set Me.Recordset = rstNwind

```

**End Sub**

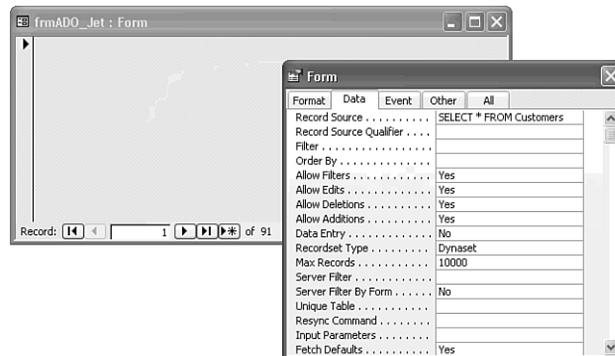
## NOTE

The preceding code includes several properties that this chapter hasn't discussed yet. The objective of this and the following sections is to get you started with a quick demonstration of the Form.Recordset property. Properties and methods of the Connection and Recordset objects are the subject of the "Exploring Top-Level ADO Properties, Methods, and Events" section that follows shortly.

-   6. Return to Access and change to Form view to execute the preceding code. Then open the Properties window and click the Data tab. Your form appears as shown in Figure 30.5, with the first of 91 records selected by the navigation buttons.

**Figure 30.5**

The frmADOTest form has its Recordset property set to an ADO DB .Recordset object opened on the Northwind.mdb Customers table.



**NOTE**

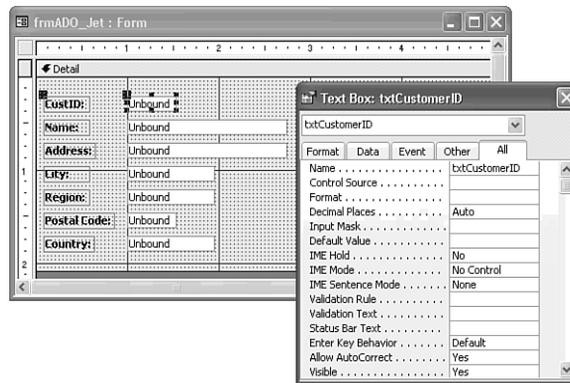
The form's Record Source property value is the SQL statement specified as the argument of the Recordset object's Open method. The Recordset Type property value appears as Dynaset, which isn't a valid ADO DB. Recordset type. The enabled Add New Record navigation button confirms that the form is updatable.

**BINDING CONTROLS TO A Recordset OBJECT WITH CODE**

Adding the equivalent of bound controls to a form whose Record Source is an ADO DB. Recordset object requires that you first add unbound controls and then bind the controls to the form's underlying Recordset with code. To create a simple data display form for the Customers table, do the following:

1. Return to Design view, display the Toolbox, and add seven unbound text boxes to the form. Name the text boxes `txtCustomerID`, `txtCompanyName`, `txtAddress`, `txtCity`, `txtRegion`, `txtPostalCode`, and `txtCountry`. Change the width of the text boxes to reflect approximately the number of characters in each of the Customer table's fields.
2. Change the label captions to **CustID:**, **Name:**, **Address:**, **City:**, **Region:**, **Postal Code:**, and **Country:**, respectively. Apply the Bold attribute to the labels for readability (see Figure 30.6).

**Figure 30.6**  
Add seven unbound text boxes to the frmADO\_Jet form.



3. To bind the Control Source property of each text box to the appropriate field of the Customers table, click the Code button and add the following lines of code immediately after the `Set Me.Recordset = rstNwind` line:

```

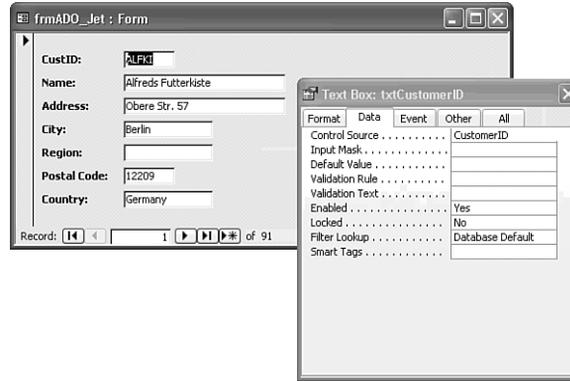
Me.txtCustomerID.ControlSource = "CustomerID"
Me.txtCompanyName.ControlSource = "CompanyName"
Me.txtAddress.ControlSource = "Address"
Me.txtCity.ControlSource = "City"
Me.txtRegion.ControlSource = "Region"
Me.txtPostalCode.ControlSource = "PostalCode"
Me.txtCountry.ControlSource = "Country"

```

4. Choose Form view and navigate the Recordset (see Figure 30.7). The Control Source property value of the text boxes displays the field name you specified in the preceding code.

**Figure 30.7**

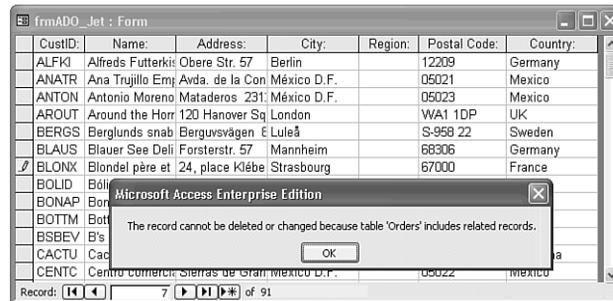
Form view displays field values in the unbound text boxes. The Data page of the Properties window of the txtCustomerID text box shows CustomerID as the Control Source property value.



5. Choose Datasheet view. The seven fields of the text boxes provide data to the columns of the datasheet, and the label captions serve as column headers.
6. Edit one of the CustID values; for example, change BLONP to BONX. If Cascade Update Related Fields for the join between the Customers and Orders tables isn't enabled, a message box states that you can't edit the field (see Figure 30.8).

**Figure 30.8**

Datasheet view of the form displays only the fields of the Customers table that have associated text boxes. Changing the value of the primary key without cascading updates displays the error message shown here.



**TIP**

To emulate a table Datasheet view with code, add to the form text boxes for every field of the table. To open a table-type ADODB.Recordset object, substitute the table name for the SQL statement as the argument of the `rstName.Open statement`. You also can specify the name of an SQL Server view or Jet QueryDef object.

## CONNECTING TO THE NORTHWINDCS MSDE DATABASE

Creating an `ADODB.Recordset` object with VBA code lets you connect to SQL Server and other client/server RDBMSs in a Jet database or Access project. To substitute the MSDE version of the Northwind sample database for `Northwind.mdb`, do the following:

1. Start your local MSDE2000 server if isn't already running.
2. Make a copy of `frmADO_Jet` as `frmADO_MSDE`, open `frmADO_MSDE` in Design view, and open the VBA Editor for `frmADO_MSDE`.
3. Delete the `.Provider = "Microsoft.Jet.OLEDB.4.0"` line. For this example, the `Open` method's argument specifies the OLE DB data provider
4. MSDE uses integrated Windows security by default, so change the `.Open CurrentProject.Path & "\Northwind.mdb", "Admin"` line to `.Open "Provider=SQLOLEDB.1;Data Source=(local);" & _ "Integrated Security=SSPI;Initial Catalog=NorthwindCS"` (SSPI is an abbreviation for Security Support Provider, Integrated.)
5. Add the following statement after the `Set Me.Recordset = rstNwind` line:  
`Me.UniqueTable = "Customers"`

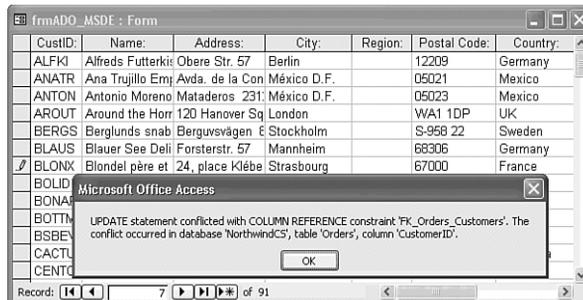
### TIP

Even if your query returns data from a single table only, you should specify the table as unique. For updatable result sets from Transact-SQL (T-SQL) queries with joins, you must set the `UniqueTable` property value to specify the "most-many" table. As an example, if your query returns values from one-to-many joins between the `Customers`, `Orders`, and `Order Details` table, `Order Details` is the "most-many" table. Fields of the `Order Details` table contribute the uniqueness to the rows of the query result set.

6. Run `frmADO_MSDE` in Datasheet view and verify that the form is updatable by temporarily editing any cell except primary-key values of the `CustID` field. You receive a constraint conflict error if you attempt to change a `CustomerID` value (see Figure 30.9).

**Figure 30.9**

When you attempt to edit a primary-key value on which other records depend, you receive the SQL Server message shown here.



**NOTE**

The ADOTest.mdb database and ADOTest.adp project in the \SEUA11\Chaptr30 folder of the accompanying CD-ROM contain the frmADO\_Jet and frmADO\_MSDE forms described in the preceding two sections. This folder contains a copy of the tables of Northwind.mdb.

## USING THE OBJECT BROWSER TO DISPLAY ADO PROPERTIES, METHODS, AND EVENTS

30

At this point in your ADO learning curve, a detailed list of properties, enumerations of constant values, methods, and events of ADO components might appear premature.

Understanding the capabilities and benefits of ADO, however, requires familiarity with ADO's repertoire of properties, methods, and events. To get the most out of ADP and to program DAP you must have a working knowledge of ADO programming techniques.

DAO objects don't fire events; ADO objects do. Access objects offer fine-grained events, but don't provide programmers with a lower-level event model for basic operations, such as connecting to a database and executing queries. Access 97's ODBCDirect offered an event model, but you couldn't bind ODBCDirect Recordsets to forms. ADO offers a complete and very fine-grained event model.



Object Browser is the most useful tool for becoming acquainted with the properties, methods, and events of ADODB objects. Object Browser also is the most convenient method for obtaining help with the syntax and usage of ADO objects, methods, and events.

**NOTE**

The examples and tabular list of properties, methods, and events of ADO objects in this and other related chapters are for ADO 2.6. Objects, methods, and events that are added by ADO 2.5+ are identified by the new in Access 2002 icon. (Access 2003's ADO 2.7 doesn't add any new elements.) If your .mdb or .adp file has a reference to ADO 2.1 or 2.5, your results for this chapter's examples might differ or fail to execute.

To use Object Browser with ADO objects, follow these steps:



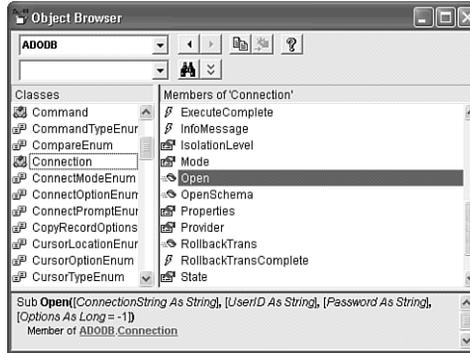
1. Open in design mode one of the forms of ADOTest.mdb that you created in the preceding sections, and then open the VBA Editor for its code. Alternatively, open the sample ADOTest.mdb or ADOTest.adp file.



2. Press F2 to open Object Browser.
3. Select ADODB in the library (upper) list.
4. Select one of the top-level components, such as `Connection`, in the Classes (left) pane.
5. Select a property, event, or method, such as `Open`, in the Members of '*ObjectName*' (right) pane. A short-form version of the syntax for the selected method or event appears in Object Browser's lower pane (see Figure 30.10).

**Figure 30.10**

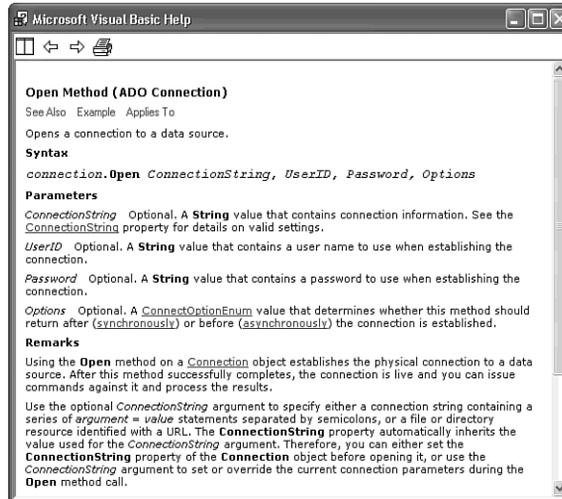
Object Browser displays in the status pane the syntax of the object class member you select in the 'ObjectName' pane.



- Click the Help button to open the help topic for the selected object, property, method, or event. Figure 30.11 shows the help topic for `ADODB.Connection.Open`.

**Figure 30.11**

The Object Browser's help button opens the online VBA help topic for the selected ADODB object, method, property, or event. The See Also link leads to related help topics. If enabled, the Example link opens a sample VBA subprocedure. The Applies To link opens a list of objects that share the method, property, or event.

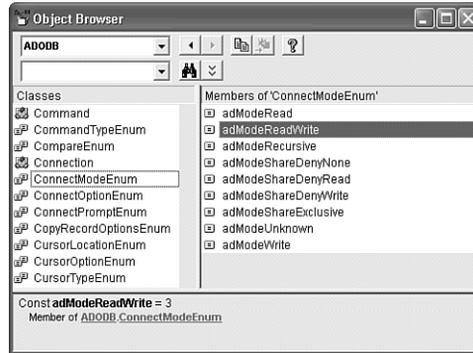


ADO type libraries also include enumerations (lists) of numeric (usually **Long**) constant values with an `ad` prefix. These constant enumerations are specific to one or more properties. Figure 30.12 shows Object Browser displaying the members of the `ConnectModeEnum` enumeration for the `Mode` property of an `ADODB.Connection` object. The lower pane displays the **Long** value of the constant.

**TIP**

You can substitute the numeric value of enumerated constants for the constant name in VBA, but doing so isn't considered a good programming practice. Numeric values of the constants might change in subsequent ADO versions, causing unexpected results when upgrading applications to a new ADO release.

**Figure 30.12**  
Object Browser displays in the status pane the numeric value of members of ADO enumerations, which are a collection of related ADO constant values.



## WORKING WITH THE ADODB.Connection OBJECT

The Connection object is the primary top-level ADO component. You must successfully open a Connection object to a data source before you can use associated Command or Recordset objects.

### CONNECTION PROPERTIES

Table 30.1 lists the names and descriptions of the properties of the ADODB.Connection object.

**TABLE 30.1** PROPERTIES OF THE ADODB.Connection OBJECT

Property Name	Data Type and Purpose
Attributes	A <b>Long</b> read/write value that specifies use of retaining transactions by the sum of two constant values. The <code>adXactCommitRetaining</code> constant starts a new transaction when calling the <code>CommitTrans</code> method; <code>adXactAbortRetaining</code> starts a new transaction when calling the <code>RollbackTrans</code> method. The default value is 0, don't use retaining transactions.
CommandTimeout	A <b>Long</b> read/write value that determines the time in seconds before terminating an <code>Execute</code> call against an associated <code>Command</code> object. The default value is 30 seconds.
ConnectionString	A <b>String</b> read/write variable that supplies specific information required by a data or service provider to open a connection to the data source.
ConnectionTimeout	A <b>Long</b> read/write value that determines the number of seconds before terminating an unsuccessful <code>Connection.Open</code> method call. The default value is 15 seconds.

*continues*

TABLE 30.1 CONTINUED

Property Name	Data Type and Purpose
CursorLocation	A <b>Long</b> read/write value that determines whether the client-side ( <code>adUseClient</code> ) or the server-side ( <code>adUseServer</code> ) cursor engine is used. The default is <code>adUseServer</code> .
DefaultDatabase	A <b>String</b> read/write variable that specifies the name of the database to use if not specified in the <code>ConnectionString</code> . For SQL Server examples, the value is the default Initial Catalog.
Errors	A pointer to the <code>Errors</code> collection for the connection that contains one or more <code>Error</code> objects if an error is encountered when attempting the connection. The later “Errors Collection and Error Objects” section describes this property.
IsolationLevel	A <b>Long</b> read/write value that determines the behavior or transactions that interact with other simultaneous transactions (see Table 30.2).
Mode	A <b>Long</b> value that determines read and write permissions for <code>Connection</code> objects (see Table 30.3).
Properties	A pointer to the OLE DB provider-specific (also called dynamic) <code>Properties</code> collection of the <code>Connection</code> object. Jet 4.0 databases have 94 <code>Property</code> objects and SQL Server databases have 93. The next section shows you how to enumerate provider-specific properties.
Provider	A <b>String</b> read/write value that specifies the name of the OLE DB data or service provider if not specified in the <code>ConnectionString</code> value. The default value is <code>MSDASQL</code> , the Microsoft OLE DB Provider for ODBC. The most common providers used in the programming chapters of this book are <code>Microsoft.Jet.OLEDB.4.0</code> , more commonly known by its code name, “Jolt 4,” and <code>SQLOLEDB</code> , the OLE DB provider for SQL Server.
State	A <b>Long</b> read-only value that specifies whether the connection to the database is open, closed, or in an intermediate state (see Table 30.4).
Version	A <b>String</b> read-only value that returns the ADO version number.

**NOTE**

Most property values identified in Table 30.1 as being read/write are writable only when the connection is in the closed state. Some provider-specific properties are read/write, but most are read-only.

**PROVIDER-SPECIFIC PROPERTIES AND THEIR VALUES**

When you’re tracking down problems with `Connection`, `Command`, `Recordset`, or `Record` objects, you might need to provide the values of some provider-specific properties to a Microsoft or another database vendor’s technical service representative. To display the

names and values of provider-specific ADODB.Property objects for an ADODB.Connection to a Jet database in the Immediate window, do the following:

1. In the declarations section of the VBA code for the frmADO\_Jet or frmADO\_MSDE form, add the following object variable declaration:

```
Private prpProp As ADODB.Property
```

Property objects exist in the Properties collection, so you don't add the **New** keyword in this case.

2. After the **End With** statement for cnnNwind, add the following instructions to print the property names and values:

```
Debug.Print cnnNwind.Properties.Count & _
    " {SQL Server|Jet} Connection Properties"
For Each prpProp In cnnNwind.Properties
    Debug.Print prpProp.Name & " = " & prpProp.Value
Next prpProp
```



3. Press Ctrl+G to open the Immediate window and delete its contents.
4. Reopen the form in Datasheet view to execute the Form\_Load event handler, and return to the VBA editor to view the result in the Immediate window (see Figure 30.13).

**Figure 30.13**

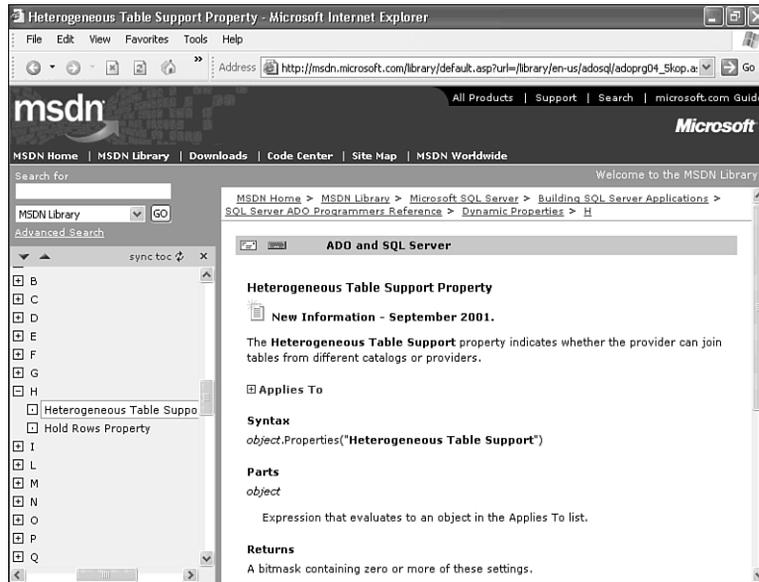
The Immediate window displays the first 19 of the 93 provider-specific properties for a connection to SQL Server 2000.

```
Immediate
93 SQL Server Connection Properties
Current Catalog = NorthwindCS
Multiple Connections = True
Reset Datasource =
Enable Fastload = False
Active Sessions = 0
Alter Column Support = 501
Asynchable Abort = False
Asynchable Commit = False
Pass By Ref Accessors = True
Catalog Location = 1
Catalog Term = database
Catalog Usage = 15
Column Definition = 1
NULL Concatenation Behavior = 1
Connection Status = 1
Data Source Name = (local)
Read-Only Data Source = False
DBMS Name = Microsoft SQL Server
DBMS Version = 08.00.0194
```



5. To find a definition of a provider-specific property of Jet or SQL Server data sources, connect to the Microsoft Web site, copy or type the name of the property in the Search For text box, add double quotes (“”) to the beginning and end of the term, and click Search. Click the appropriate link (usually the first) to display the definition of the property (see Figure 30.14).
6. After you've satisfied your curiosity about provider-specific properties and their values, comment out or delete the added code. Sending a significant amount of data to the Immediate window delays opening the form.

**Figure 30.14**  
Most Jet and SQL Server provider-specific properties have pages in the Microsoft Developer Network (MSDN) library for the Platform SDK (psdk). Click Show TOC to find your relative location within the library.



## NOTE



Pages that define SQL Server-specific properties specify values by reference to `DBPROP_VAL_...` constants whose values aren't included in the table. Many searches for Jet-specific Property object definitions lead to the "Appendix C: Microsoft Jet 4.0 OLE DB Provider-Defined Property Values" page ([http://msdn.microsoft.com/library/techart/daotoadoupdate\\_topic15.htm](http://msdn.microsoft.com/library/techart/daotoadoupdate_topic15.htm)), which provides a set of constant values that you can add to an Access module.

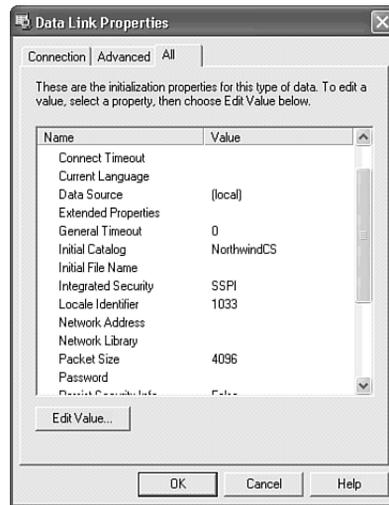
Some of the SQL Server provider-specific properties appear in a list on the All page of the Data Link Properties dialog for a project's connection. To view these properties and their values, when set, open the NorthwindCS project, choose **File**, **Connection** to open the Data Link Properties dialog, and click the All tab (see Figure 30.15).

## TIP



The "Appendix A: DAO to ADO Quick Reference" page ([http://msdn.microsoft.com/library/techart/daotoadoupdate\\_topic13.htm](http://msdn.microsoft.com/library/techart/daotoadoupdate_topic13.htm)) of the "Migrating from DAO to ADO" white paper contains a table that translates DAO objects and properties to ADO objects, properties, and provider-specific Jet properties. To create an easily searchable version, copy the table to the clipboard, paste it into a Word document and save the file in both .doc and .htm formats. Importing the .htm table to a Jet or SQL Server table lets you view the contents in a searchable datasheet (see Figure 30.16). Contents of the Microsoft Web site are copyrighted, so the table isn't included in this chapter's example databases.

**Figure 30.15**  
The All page of the Data Link Properties dialog for the NorthwindCS connection to MSDE displays a few of the 93 provider-specific properties of the OLE DB Provider for SQL Server (SQLOLEDB).



**Figure 30.16**  
This Jet table was created by importing a copy of the HTML table of the “DAO to ADO Cross Reference” page from the Microsoft Web site. Property or method names with number suffixes, such as DefaultType1, refer to footnotes in the source Web page.

ID	DAO Object	DAO Property/Method	ADO/ADOX/JRO	Object	ADO Property/Method
1	DBEngine	DefaultType1	N/A	N/A	N/A
2	DBEngine	DefaultPassword1	N/A	N/A	N/A
3	DBEngine	DefaultUser1	N/A	N/A	N/A
4	DBEngine	IniPath	ADO	Connection	Jet OLEDB:Registry Path2
5	DBEngine	LoginTimeout	ADO	Connection	ConnectionTimeout
6	DBEngine	SystemDB	ADO	Connection	Jet OLEDB:System Database2
7	DBEngine	Version	ADO	Connection	Version
8	DBEngine	BeginTrans	ADO	Connection	BeginTrans
9	DBEngine	CommitTrans	ADO	Connection	CommitTrans
10	DBEngine	Rollback	ADO	Connection	RollbackTrans
11	DBEngine	CompactDatabase	JRO	JetEngine	CompactDatabase
12	DBEngine	CreateDatabase	ADOX	Catalog	Create
13	DBEngine	CreateWorkspace	ADO	Connection	Open
14	DBEngine	Idle	JRO	JetEngine	RefreshCache
15	DBEngine	OpenDatabase	ADO	Connection	Open
16	DBEngine	RegisterDatabase1	N/A	N/A	N/A
17	DBEngine	RepairDatabase1	N/A	N/A	N/A
18	DBEngine	SetOption	ADO	Connection	Properties3
19	Workspace	IsolateODBCTrans	ADO	Connection	Isolation Levels2
20	Workspace	LoginTimeout	ADO	Connection	ConnectionTimeout
21	Workspace	Name1	N/A	N/A	N/A

## TRANSACTION ISOLATION LEVELS

The ability to specify the transaction isolation level applies only when you use the `BeginTrans`, `CommitTrans`, and `RollbackTrans` methods (see Table 30.6 later in this chapter) to perform a transaction on a `Connection` object. If multiple database users simultaneously execute transactions, your application should specify how it responds to other transactions in-process. Table 30.2 lists the options for the degree of your application’s isolation from other simultaneous transactions.

**TABLE 30.2** CONSTANT ENUMERATION FOR THE *IsolationLevel* PROPERTY

<b>IsolationLevelEnum</b>	<b>Description</b>
adXactCursorStability	Allows reading only committed changes in other transactions (default value).
adXactBrowse	Allows reading uncommitted changes in other transactions.
adXactChaos	The transaction won't overwrite changes made to transaction(s) at a higher isolation level.
adXactIsolated	All transactions are independent of (isolated from) other transactions.
adXactReadCommitted	Same as adXactCursorStability.
adXactReadUncommitted	Same as adXactBrowse.
adXactRepeatableRead	Prohibits reading changes in other transactions.
adXactSerializable	Same as adXactIsolated.
adXactUnspecified	The transaction level of the provider can't be determined.

**NOTE**

Enumeration tables in this book list the default value first, followed by the remaining constants in alphabetical order. Where two members of Table 30.2 represent the same isolation level, one of the members is included for backward compatibility.

**THE *Connection.Mode* PROPERTY**

Unless you have a specific reason to specify a particular `ADODB.Connection.Mode` value, the default `adModeUnknown` is adequate. The Jet OLE DB provider defaults to `adModeShareDenyNone`. The Access Permissions list on the Advanced page of the Data Link properties page for `SQLOLEDB` is disabled, but you can set the `Mode` property with code. Table 30.3 lists all the constants for the `Mode` property.

**TABLE 30.3** CONSTANT ENUMERATION FOR THE *Mode* PROPERTY

<b>ConnectModeEnum</b>	<b>Description</b>
adModeUnknown	No connection permissions have been set on the data source (default value).
adModeRead	Connect with read-only permission.
adModeReadWrite	Connect with read/write permissions.
adModeRecursive	If an <code>adModeShareDeny...</code> flag is specified, applies the mode to child records of a chaptered (hierarchical) <code>Recordset</code> object.

TABLE 30.3 CONTINUED

ConnectModeEnum	Description
adoModeRecursive	Used in conjunction with the Record objects, which this chapter doesn't cover.
adModeShareDenyNone	Don't deny other users read or write access.
adModeShareDenyRead	Deny others permission to open a read connection to the data source.
adModeShareDenyWrite	Deny others permission to open a write connection to the data source.
adModeShareExclusive	Open the data source for exclusive use.
adModeWrite	Connect with write-only permission.

**TIP**

You often can improve performance of client/server decision-support applications by opening the connection as read only (`adModeRead`). Modifying the structure of a database with SQL's DDL usually requires exclusive access to the database (`adModeShareExclusive`).

**THE Connection.State PROPERTY**

Table 30.4 lists the constants that return the state of the Connection object. These constants also are applicable to the State property of the Command and Recordset objects.

It's common to open and close connections as needed to reduce the connection load on the database. (Each open connection to a client/server database consumes a block of memory.) In many cases, you must test whether the Connection object is open or closed before applying the Close or Open method, or changing Connection property values, which are read-only when the connection is open.

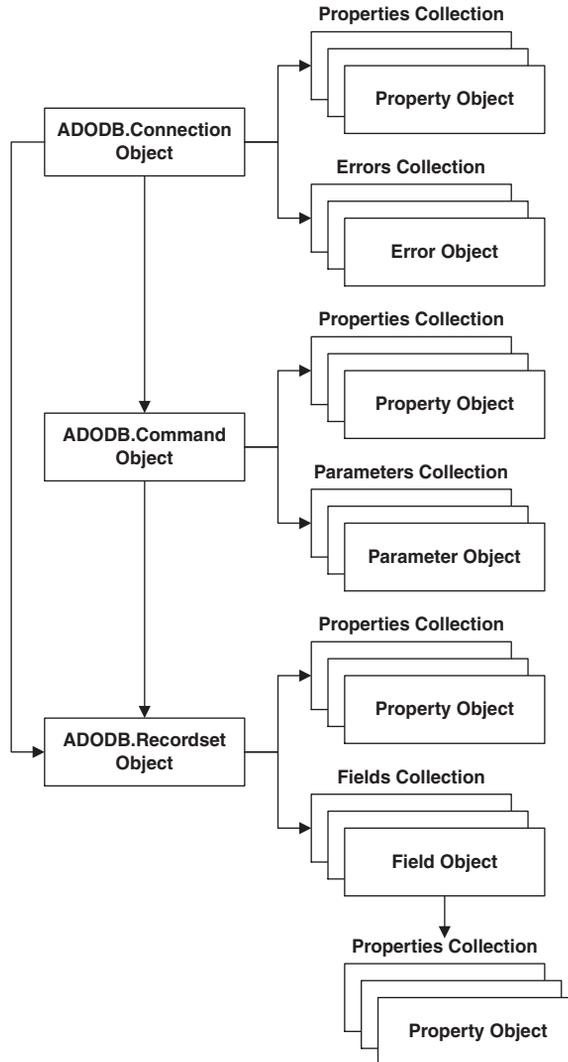
TABLE 30.4 CONSTANT ENUMERATION FOR THE State PROPERTY

ObjectStateEnum	Description
adStateClosed	The Connection (or other object) is closed (default value).
adStateConnecting	A connection to the data source is in progress.
adStateExecuting	The Execute method of a Connection or Command object has been called.
adStateFetching	Rows are returning to a Recordset object.
adStateOpen	The Connection (or other object) is open (active).

### Errors COLLECTION AND Error OBJECTS

Figure 30.17 illustrates the relationship between top-level ADO components and their collections. The dependent Errors collection is a property of the Connection object, and if errors are encountered with any operation on the connection, contains one or more Error objects. The Errors collection has one property, Count, which you test to determine whether an error has occurred after executing a method call on Connection and Recordset objects. A collection is required, because it's possible for an object to generate several errors.

**30** **Figure 30.17**  
 The Connection, Command, and Recordset objects have Properties and Errors collections. The Command object also has a Parameters collection and the Recordset object has a Fields Collection. The new Record object isn't included in this diagram.



The Errors collection has two methods, Clear and Item. The Clear method deletes all current Error objects in the collection, resetting the value of Count to 0. The Item method,

which is the default method of the Errors and other collections, returns an object reference (pointer) to an Error object. The syntax for explicit and default use of the Item method is

```
Set errName = cnnName.Errors.Index({strName|intIndex})
Set errName = cnnName.Errors({strName|intIndex})
```

The Error object has the seven read-only properties listed in Table 30.5. Error objects have no methods or events. The InfoMessage event of the Connection object, described in the “Connection Events” section later in this chapter, fires when an Error object is added to the Errors collection and supplies a pointer to the newly added Error object.

**TABLE 30.5** PROPERTY NAMES AND DESCRIPTIONS OF THE Error OBJECT

Property Name	Description
Description	A <b>String</b> value containing a brief text description of the error
HelpContext	A <b>Long</b> value specifying the error’s context ID in a Windows Help file
HelpFile	A <b>String</b> value specifying the full path to and name of the Windows Help file, usually for the data provider
NativeError	A <b>Long</b> value specifying a provider-specific error code
Number	A <b>Long</b> value specifying the number assigned to the error by the provider or data source
Source	A <b>String</b> value containing the name of the object that generated the error, ADODB. <i>ObjectName</i> for ADO errors
SQLState	A <b>String</b> value (SQLSTATE) containing a five-letter code specified by the ANSI/ISO SQL-92 standard, consisting of two characters specifying Condition, followed by three characters for Subcondition

→ For the basics of error handling in VBA, see “Handling Runtime Errors,” p. 1176.

**NOTE**

Unfortunately, not all RDBMS vendors implement SQLSTATE in the same way. If you test the SQLState property value, make sure to follow the vendor-specific specifications for Condition and Subcondition values.

Listing 30.1 is an example of code to open a Connection (cnnNwind) and a Recordset (rstCusts) with conventional error handling; rstCusts supplies the Recordset property of the form. The “Non-existent” table name generates a “Syntax error in FROM clause” error in the Immediate window. The `Set ObjectName = Nothing` statements in the error handler recover the memory consumed by the objects.

## LISTING 30.1 VBA CODE THAT WRITES ERROR PROPERTIES TO THE IMMEDIATE WINDOW

```

Private Sub Form_Load
    Dim cnnNwind As New ADODB.Connection
    Dim rstCusts As New ADODB.Recordset

    On Error GoTo CatchErrors
    cnnNwind.Provider = "Microsoft.Jet.OLEDB.4.0"
    cnnNwind.Open CurrentProject.Path & "\Northwind.mdb", "Admin"
    With rstCusts
        Set .ActiveConnection = cnnNwind
        .CursorType = adOpenKeyset
        .LockType = adLockBatchOptimistic
        .Open "SELECT * FROM Non-existent"
    End With
    Set Me.Recordset = rstCusts
Exit Sub

CatchErrors:
    Dim colErrors As Errors
    Dim errNwind As Error
    Set colErrors = cnnNwind.Errors
    For Each errNwind In colErrors
        Debug.Print "Description: " & errNwind.Description
        Debug.Print "Native Error: " & errNwind.NativeError; ""
        Debug.Print "SQL State: " & errNwind.SQLState
        Debug.Print vbCrLf
    Next errNwind
    Set colErrors = Nothing
    Set errNwind = Nothing
    Set rstCusts = Nothing
    Set cnnNwind = Nothing
Exit Sub
End Sub

```



## NOTE

The frmErrors form of ADOTest.mdb and ADOTest.adp incorporates the preceding code. Open the form to execute the code, change to Design view, open the VBA editor, and press Ctrl+G to read the error message in the Immediate window.

## CONNECTION METHODS

Table 30.6 lists the methods of the ADODB.Connection object. Only the Execute, Open, and OpenSchema methods accept argument values. The OpenSchema method is of interest primarily for creating database diagrams, data transformation for data warehouses and marts, and online analytical processing (OLAP) applications.

**TABLE 30.6** METHODS OF THE ADODB.Connection OBJECT

Method	Description
BeginTrans	Initiates a transaction; must be followed by CommitTrans and/or RollbackTrans.
Close	Closes the connection.
CommitTrans	Commits a transaction, making changes to the data source permanent. (Requires a prior call to the BeginTrans method.)
Execute	Returns a forward-only Recordset object from a SELECT SQL statement. Also used to execute statements that don't return Recordset objects, such as INSERT, UPDATE, and DELETE queries or DDL statements. You use this method to execute T-SQL stored procedures, regardless of whether they return a Recordset.
Open	Opens a connection based on a connection string.
OpenSchema	Returns a Recordset object that provides information on the structure of the data source, called metadata.
RollbackTrans	Cancels a transaction, reversing any temporary changes made to the data source. (Requires a prior call to the BeginTrans method.)

### THE Connection.Open AND Connection.OpenSchema METHODS

You must open a connection before you can execute a statement on it. The syntax of the Open method is

```
cnnName.Open [strConnect[, strUID[, strPwd, lngOptions]]]
```

Alternatively, you can assign the connection string values to the Connection object's Provider and ConnectionString properties. The following example, similar to that for the Recordset object examples early in the chapter, is for a connection to Northwind.mdb in the same folder as the application .mdb:

```
With cnnNwind
    .Provider = "Microsoft.Jet.OLEDB.4.0"
    .ConnectionString = CurrentProject.Path & "\Northwind.mdb"
    .Open
End With
```

In this case, all the information required to open a connection to Northwind.mdb is provided as property values, so the Open method needs no argument values.

If you're creating a data dictionary or designing a generic query processor for a client/server RDBMS, the OpenSchema method is likely to be of interest to you. Otherwise, you might want to skip the details of the OpenSchema method, which is included here for completeness. Schema information is called *metadata*, data that describes the structure of data.

**TIP**

ADOX 2.7 defines a `Catalog` object for Jet 4.0 databases that's more useful for Jet databases than the generic `OpenSchema` method, which is intended primarily for use with client/server RDBMs. The `Catalog` object includes `Groups`, `Users`, `Tables`, `Views`, and `Procedures` collections.

**THE Connection.Execute METHOD**

The syntax of the `Connection.Execute` method to return a reference to a forward-only `ADODB.Recordset` object is

```
Set rstName = cnnName.Execute (strCommand, [lngRowsAffected[, lngOptions]])
```

Alternatively, you can use named arguments for all ADO methods. Named arguments, however, require considerably more typing than conventional comma-separated argument syntax. The named argument equivalent of the preceding `Set` statement is

```
Set rstName = cnnName.Execute (Command:=strCommand, _  
    RowsAffected:=lngRowsAffected, Options:=lngOptions)
```

If `strCommand` doesn't return a `Recordset`, the syntax is

```
cnnName.Execute strCommand, [lngRowsAffected[, lngOptions]]
```

The value of `strCommand` can be an SQL statement, a table name, the name of a stored procedure, or an arbitrary text string acceptable to the data provider.

**TIP**

For best performance, specify a value for the `lngOptions` argument (see Table 30.7) so the provider doesn't need to interpret the statement to determine its type. The optional `lngRowsAffected` argument returns the number of rows affected by an `INSERT`, `UPDATE`, or `DELETE` query; these types of queries return a closed `Recordset` object. A `SELECT` query returns 0 to `lngRowsAffected` and an open, forward-only `Recordset` with 0 or more rows. The value of `lngRowsAffected` is 0 for T-SQL updates queries and stored procedures that include the `SET NOCOUNT ON` statement.

**TABLE 30.7** CONSTANT ENUMERATION FOR THE `lngOptions` ARGUMENT OF THE `Execute` METHOD FOR `Connection` AND `Command` OBJECTS

<b>CommandTypeEnum</b>	<b>Description</b>
<code>adCmdUnknown</code>	The type of command isn't specified (default). The data provider determines the syntax of the command.
<code>adCmdFile</code>	The command is the name of a file in a format appropriate to the object type.
<code>adCmdStoredProc</code>	The command is the name of a stored procedure.
<code>adCmdTable</code>	The command is a table name, which generates an internal <code>SELECT * FROM TableName</code> query.

CommandTypeEnum	Description
adCmdTableDirect	The command is a table name, which retrieves rows directly from the table
adCmdText	The command is an SQL statement.

Forward-only Recordset objects, created by what's called a firehose cursor, provide the best performance and minimum network traffic in a client/server environment. However, forward-only Recordsets are limited to manipulation by VBA code. If you set the RecordSource property of a form to a forward-only Recordset, controls on the form don't display field values.

## CONNECTION EVENTS

Events are useful for trapping errors, eliminating the need to poll the values of properties, such as State, and performing asynchronous database operations. To expose the ADODB.Connection events to your application, you must use the **WithEvents** reserved word (without **New**) to declare the ADODB.Connection object in the Declarations section of a class or form module and then use a **Set** statement with **New** to create an instance of the object, as shown in the following example:

```
Private WithEvents cnnName As ADODB.Connection

Private Sub Form_Load
    Set cnnName = New ADODB.Connection
    ...
    Code using the Connection object
    ...
    cnnName.Close
End Sub
```

The preceding syntax is required for most Automation objects that source (expose) events. Event-handling subprocedures for Automation events often are called event sinks. Source and sink terminology derives from the early days of transistors; the source (emitter) supplies electrons and the sink (collector) accumulates electrons.

Table 30.8 lists the events that appear in the Procedures list of the code-editing window for the *cnnName* Connection object and gives a description of when the events fire.

**TABLE 30.8** EVENTS FIRED BY THE ADODB.Connection OBJECT

Event Name	When Fired
BeginTransComplete	After the BeginTrans method executes
CommitTransComplete	After the CommitTrans method executes
ConnectComplete	After a Connection to the data source succeeds
Disconnect	After a Connection is closed

*continues*

TABLE 30.8 CONTINUED

Event Name	When Fired
ExecuteComplete	On completion of the <code>Connection.Execute</code> or <code>Command.Execute</code> method call
InfoMessage	When an Error object is added to the <code>ADODB.Connection.Errors</code> collection
RollbackTransComplete	After the <code>RollbackTrans</code> method executes
WillConnect	On calling the <code>Connection.Open</code> method but before the connection is made
WillExecute	On calling the <code>Connection.Execute</code> or <code>Command.Execute</code> method, just before the command executes a connection

**TIP**

Take full advantage of ADO events in your VBA data-handling code. Relatively few developers currently use event-handling code in ordinary database front ends. ADO's event model will be of primary interest to developers migrating from Access 97's RDO to ADO. Developers of data warehousing and OLAP applications, which often involve very long-running queries, are most likely to use events in conjunction with asynchronous query operations.

## USING THE `ADODB.Command` OBJECT

The primary purpose of the `Command` object is to execute parameterized stored procedures, either in the form of the default temporary prepared statements or persistent, precompiled T-SQL statements in SQL Server databases. MSDE and SQL Server create temporary prepared statements that exist only for the lifetime of the current client connection. Precompiled SQL statements are procedures stored in the database file; their more common name is stored procedure. When creating `Recordset` objects from ad hoc SQL statements, the more efficient approach is to bypass the `Command` object and use the `Recordset.Open` method.

### Command PROPERTIES

The `Command` object has relatively few properties, many of which duplicate those of the `Connection` object. Table 30.9 lists the names and descriptions of the `Command` object's properties. Like the `Connection` object, the `Command` object has its own provider-specific `Properties` collection, which you can print to the Immediate window using statements similar to those for `Command` objects described in the earlier "Provider-Specific Properties and Their Values" section.

**TIP**

The `Command` object is required to take advantage of ADO 2.6+'s `Stream` object, which contains data in the form of a continuous stream of binary data or text. Text streams often contain XML documents or document fragments returned from SQL Server 2000 XML AUTO queries. The Microsoft OLE DB Provider for Internet Publishing (MSDAIPP) enables `Connection`, `Recordset`, `Record`, and `Stream` objects to bind to a URL and retrieve data into a `Stream` object. Windows XP/2000+'s Internet Information Server (IIS) 5.0+ adds the MSDAIPP provider.

TABLE 30.9 PROPERTIES OF THE `Command` OBJECT

Property Name	Description
<code>ActiveConnection</code>	A pointer to the <code>Connection</code> object associated with the <code>Command</code> . Use <code>Set cmdName.ActiveConnection = cnnName</code> for an existing open <code>Connection</code> . Alternatively, you can use a valid connection string to create a new connection without associating a named <code>Connection</code> object. The default value is <b>Null</b> .
 <code>CommandStream</code>	A <b>Variant</b> read/write value that contains the input stream used to specify the output stream.
<code>CommandText</code>	A <b>String</b> read/write value that specifies an SQL statement, table name, stored procedure name, or an arbitrary string acceptable to the provider of the <code>ActiveConnection</code> . The value of the <code>CommandType</code> property determines the format of the <code>CommandText</code> value. The default value is an empty string, "" . <code>CommandText</code> and <code>CommandStream</code> are mutually exclusive. You can't specify a <code>CommandStream</code> and a <code>CommandText</code> value for the same <code>Command</code> object.
<code>CommandTimeout</code>	A <b>Long</b> read/write value that determines the time in seconds before terminating a <code>Command.Execute</code> call. This value overrides the <code>Connection.CommandTimeout</code> setting. The default value is 30 seconds.
<code>CommandType</code>	A <b>Long</b> read/write value that specifies how the data provider interprets the value of the <code>CommandText</code> property. ( <code>CommandType</code> is the equivalent of the optional <code>lngCommandType</code> argument of the <code>Connection.Execute</code> method, described earlier in the chapter (refer to Table 30.7). The default value is <code>adCmdUnknown</code> .)

*continues*

TABLE 30.9 CONTINUED

Property Name	Description
 Dialect	A <b>String</b> read/write value that accepts one of four globally unique ID (GUID) values specifying the type of <code>CommandStream</code> object. Valid settings are <code>DBGUID_DEFAULT</code> (the provider decides how to handle the <code>CommandStream</code> value), <code>DBGUID_SQL</code> (an SQL statement), <code>DBGUID_MSSQLXML</code> (an SQL Server XML AUTO query), and <code>DBGUID_XPATH</code> (an SQL Server XPath query). The values of these constants are defined in the “Programming Stream Objects” section near the end of this chapter.
Name	A <b>String</b> read/write value specifying the name of the command, such as <code>cmdNwind</code> .
 NamedParameters	A <b>Boolean</b> read/write value that, when set to <b>True</b> , specifies that the names of members of the <code>Parameters</code> collection be used, rather than their sequence, when passing parameter values to and from SQL Server functions and stored procedures, or accepting return or output values from stored procedures.
Prepared	A <b>Boolean</b> read/write value that determines whether the data source compiles the <code>CommandText</code> SQL statement as a prepared statement (a temporary stored procedure). The prepared statement exists only for the lifetime of the <code>Command</code> object’s <code>ActiveConnection</code> . Many client/server RDBMSs, including Microsoft SQL Server, support prepared statements. If the data source doesn’t support prepared statements, setting <code>Prepared</code> to <b>True</b> results in a trappable error.
Properties	Same as the <code>Properties</code> collection of the <code>Connection</code> object.
State	A <b>Long</b> read/write value specifying the status of the <code>Command</code> . Refer to Table 30.4 for <code>ObjectStateEnum</code> constant values.

**TIP**

Always set the `CommandType` property to the appropriate `adCmd...` constant value. If you accept the default `adCmdUnknown` value, the data provider must test the value of `CommandText` to determine whether it is the name of a stored procedure, a table, or an SQL statement before executing the query. If the targeted database contains a large number of objects, testing the `CommandText` value for each `Command` object you execute can significantly reduce performance.

The initial execution of a prepared statement often is slower than for a conventional SQL query because some data sources must compile, rather than interpret, the statement. Thus, you should limit use of prepared statements to parameterized queries in which the query is executed multiple times with differing parameter values.

Parameters COLLECTION

To supply and accept parameter values, the Command object uses the Parameters collection, which is similar to the DAO and ODBCDirect Parameters collections. ADODB.Parameters is independent of its parent, ADODB.Command, but you must associate the Parameters collection with a Command object before defining or using Parameter objects.

The Parameters collection has a read-only Long property, Count, an Item property that returns a Parameter object, and the methods listed in Table 30.10. The syntax for the Count and Item properties property is

```
lngNumParms = cmmName.Parameters.Count
prmParamName = cmmName.Parameters.Item(lngIndex)
```

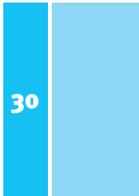


TABLE 30.10 METHOD NAMES, DESCRIPTIONS, AND CALLING SYNTAX FOR THE Parameters COLLECTION

Method Name	Description and VBA Calling Syntax
Append	Appends a Parameter object created by the cmmName.CreateParameter method, described in the “Command Methods” section, to the collection. The calling syntax is Parameters.Append prmName.
Delete	Deletes a Parameter object from the collection. The calling syntax is cmmName.Parameters.Delete {strName intIndex}, where strName is the name of the Parameter or intIndex is the 0-based ordinal position (index) of the Parameter in the collection.
Refresh	Retrieves the properties of the current set of parameters for the stored procedure or query specified as the value of the CommandText property. The calling syntax is cmmName.Parameters.Refresh. If you don't specify your own members of the Parameters collection with the CreateParameter method, accessing any member of the Parameters collection automatically calls the Refresh method. If you apply the Refresh method to a data source that doesn't support stored procedures, prepared statements, or parameterized queries, the Parameters collection is empty (cmmName.Parameters.Count = 0).

You gain a performance improvement for the initial execution of your stored procedure or query if you use the cmmName.CreateParameter method to predefine the required Parameter objects. The Refresh method makes a round-trip to the server to retrieve the properties of each Parameter.

Parameter OBJECT

One Parameter object must exist in the Parameters collection for each parameter of the stored procedure, prepared statement, or parameterized query. Table 30.11 lists the property names and descriptions of the Parameter object. The syntax for getting and setting Parameter property values is

```
typPropValue = cmmName.Parameters({strName!lngIndex}).PropertyName
cmmName.Parameters({strName!lngIndex}).PropertyName = typPropValue
```

You don't need to use the `Index` property of the `Parameters` collection; `Index` is the default property of `Parameters`.

**TABLE 30.11** PROPERTY NAMES AND DESCRIPTIONS FOR `Parameter` OBJECTS

Property Name	Description
Attributes	A <b>Long</b> read/write value representing the sum of the <code>adParam...</code> constants listed in Table 30.12.
Direction	A <b>Long</b> read/write value representing one of the <code>adParam...</code> constants listed in Table 30.13.
Name	A <b>String</b> read/write value containing the name of the <code>Parameter</code> object, such as <code>prmStartDate</code> . The name of the <code>Parameter</code> object need not (and usually does not) correspond to the name of the corresponding parameter variable of the stored procedure. After the <code>Parameter</code> is appended to the <code>Parameters</code> collection, the <code>Name</code> property value is read-only.
NumericScale	A <b>Byte</b> read/write value specifying the number of decimal places for numeric values.
Precision	A <b>Byte</b> read/write value specifying the total number of digits (including decimal digits) for numeric values.
Size	A <b>Long</b> read/write value specifying the maximum length of variable-length data types supplied as the <code>Value</code> property. You must set the <code>Size</code> property value before setting the <code>Value</code> property to variable-length data.
Type	A <b>Long</b> read/write value representing a valid OLE DB 2+ data type, the most common of which are listed in Table 30.14.
Value	The value of the parameter having a data type corresponding to the value of the <code>Type</code> property.

**TABLE 30.12** CONSTANT VALUES FOR THE `Attributes` PROPERTY OF THE `Parameter` OBJECT

ParameterAttributesEnum	Description
<code>adParamSigned</code>	The <code>Parameter</code> accepts signed values (default).
<code>adParamNullable</code>	The <code>Parameter</code> accepts <b>Null</b> values.
<code>adParamLong</code>	The <code>Parameter</code> accepts long binary data.

**TABLE 30.13** CONSTANT VALUES FOR THE `Direction` PROPERTY OF THE `Parameter` OBJECT

<code>ParameterDirectionEnum</code>	Description
<code>adParamInput</code>	Specifies an input parameter (default).
<code>adParamOutput</code>	Specifies an output parameter.
<code>adParamInputOutput</code>	Specifies an input/output parameter.
<code>adParamReturnValue</code>	Specifies the return value of a stored procedure.
<code>adParamUnknown</code>	The parameter direction is unknown.

The `Type` property has the largest collection of constants of any ADO enumeration; you can review the entire list of data types by selecting the `DataTypeEnum` class in Object Browser. Most of the data types aren't available to VBA programmers, so Table 30.14 shows only the most commonly used `DataTypeEnum` constants. In most cases, you only need to choose among `adChar` (for **String** values), `adInteger` (for **Long** values), and `adCurrency` (for **Currency** values). You use the `adDate` data type to pass Date/Time parameter values to Jet databases, but not to most stored procedures. Stored procedures generally accept `datetime` parameter values as the `adChar` data type, with a format, such as `mm/dd/yyyy`, acceptable to the RDBMS.

**TABLE 30.14** COMMON CONSTANT VALUES FOR THE `Type` PROPERTY OF THE `Parameter` AND `Field` OBJECTS

<code>DataTypeEnum</code>	Description of Data Type
<code>adBinary</code>	<b>Binary</b> value.
<code>adBoolean</code>	<b>Boolean</b> value.
<code>adChar</code>	<b>String</b> value.
<code>adCurrency</code>	<b>Currency</b> values are fixed-point numbers with four decimal digits stored in an 8-byte, signed integer, which is scaled (divided) by 10,000.
<code>adDate</code>	<b>Date</b> values are stored as a <b>Double</b> value, the integer part being the number of days since December 30, 1899, and the decimal part being the fraction of a day.
<code>adDecimal</code>	Exact numeric value with a specified precision and scale.
<code>adDouble</code>	<b>Double</b> -precision floating-point value.
<code>adInteger</code>	4-byte signed <b>Long</b> integer.
<code>adLongVarBinary</code>	Long binary value (Parameter objects only).
<code>adLongVarChar</code>	<b>String</b> value greater than 225 characters (Parameter objects only).

*continues*

TABLE 30.14 CONTINUED

DataTypeEnum	Description of Data Type
adNumeric	Exact numeric value with a specified precision and scale.
adSingle	<b>Single</b> -precision floating-point value.
adSmallInt	2-byte signed <b>Integer</b> .
adTinyInt	<b>Byte</b> (1-byte signed integer).
adVarBinary	Binary value for Jet OLE Object and SQL Server image fields (Parameter objects only).
adVarChar	<b>String</b> value for Jet Memo and SQL Server text fields (Parameter objects only).

**NOTE**

The values for the `Type` property in the preceding table are valid for the `Type` property of the `Field` object, discussed later in the chapter, except for those data types in which “Parameter objects only” appears in the Description of Data Type column. The members of `DataTypeEnum` are designed to accommodate the widest possible range of desktop and client/server RDBMSs, but the `ad...` constant names are closely related to those for the field data types of Microsoft SQL Server 2000 and MSDE, which support Unicode strings.

For a complete list with descriptions of `DataTypeEnum` constants, go to <http://msdn.microsoft.com/library/en-us/ado270/html/mdcstdatatypeenum.asp>.

The `Parameter` object has a single method, `AppendChunk`, which you use to append long text (`adLongText`) or long binary (`adLongVarBinary`) **variant** data as a parameter value. The syntax of the `AppendChunk` method call is

```
cmmName.Parameters({strName|lngIndex}).AppendChunk = varChunk
```

The `adParamLong` flag of the `prmName.Attributes` property must be set to apply the `AppendChunk` method. If you call `AppendChunk` more than once on a single `Parameter`, the second and later calls append the current value of `varChunk` to the parameter value.

**Command METHODS**

Command objects have only three methods: `Cancel`, `CreateParameter` and `Execute`. Executing `Command.Cancel` terminates an asynchronous command opened with the `adAsyncConnect`, `adAsyncExecute`, or `adAsyncFetch` option.

You must declare an `ADODB.Parameter` object, `prmName`, prior to executing `CreateParameter`. The syntax of the `CreateParameter` method call is

```
Set prmName = cmmName.CreateParameter [strName[, lngType[, _
    lngDirection[, lngSize[, varValue]]]]]
cmmName.Parameters.Append prmName
```

The arguments of `CreateParameter` are optional only if you subsequently set the required Parameter property values before executing the `Command`. For example, if you supply only the `strName` argument, you must set the remaining properties, as in the following example:

```
Set prmName = cmmName.CreateParameter strName
cmmName.Parameters.Append prmName
With prmName
    .Type = adChar
    .Direction = adParamInput
    .Size = Len(varValue)
    .Value = varValue
End With
```

The syntax of the `Command.Execute` method is similar to that for the `Connection.Execute` method except for the argument list. The following syntax is for `Command` objects that return `Recordset` objects:

```
Set rstName = cmmName.Execute([lngRowsAffected[, _
    avarParameters[, lngOptions]])
```

For `Command` objects that don't return rows, use this form:

```
cmmName.Execute [lngRowsAffected[, avarParameters[, lngOptions]]]
```

All the arguments of the `Execute` method are optional if you set the required `Command` property values before applying the `Execute` method. Listing 30.2 later in this chapter gives an example of the use of the `Command.Execute` method without arguments.

#### TIP

Presetting all property values of the `Command` object, rather than supplying argument values to the `Execute` method, makes your VBA code easier for others to comprehend.

Like the `Connection.Execute` method, the returned value of `lngRowsAffected` is 0 for `SELECT` and `DDL` queries and the number of rows modified by execution of `INSERT`, `UPDATE`, and `DELETE` queries. (For SQL Server, `lngRowsAffected` is 0 if the SQL statement includes `SET NOCOUNT ON`.) The `avarParameters` argument is an optional **variant** array of parameter values. Using the `Parameters` collection is a better practice than using the `avarParameters` argument because output parameters don't return correct values to the array. For `lngOptions` constant values, refer to Table 30.7.

## CODE TO PASS PARAMETER VALUES TO A STORED PROCEDURE

Most stored procedures that return `Recordset` objects require input parameters to supply values to `WHERE` clause criteria to limit the number of rows returned. The code of Listing 30.2 executes a simple SQL Server 2000 stored procedure with a `Command` object. The `Sales by Year` stored procedure of the `NorthwindCS` project has two `datetime` input parameters, `@Beginning_Date` and `@Ending_Date`, the values for which are supplied by `strBegDate` and `strEndDate`, respectively. The stored procedure, whose SQL statement follows, returns the `ShippedDate` and `OrderID` columns of the `Orders` table, the `Subtotal` column of the `Order`

Subtotals view, and a calculated Year value. The stored procedure returns rows for values of the OrderDate field between strBegDate and strEndDate.

```
ALTER PROCEDURE "Sales by Year"
    @Beginning_Date datetime,
    @Ending_Date datetime
AS SELECT Orders.ShippedDate, Orders.OrderID,
    "Order Subtotals".Subtotal,
    DATENAME(yy,ShippedDate) AS Year
FROM Orders INNER JOIN "Order Subtotals"
    ON Orders.OrderID = "Order Subtotals".OrderID
WHERE Orders.ShippedDate Between @Beginning_Date And @Ending_Date
```

### LISTING 30.2 CODE USING A Command OBJECT TO EXECUTE A PARAMETERIZED STORED PROCEDURE

```
Option Explicit
Option Compare Database

Private cnnOrders As New ADODB.Connection
Private cmmOrders As New ADODB.Command
Private prmBegDate As New ADODB.Parameter
Private prmEndDate As New ADODB.Parameter
Private rstOrders As New ADODB.Recordset

Private Sub Form_Load()
    Dim strBegDate As String
    Dim strEndDate As String
    Dim strFile As String

    strBegDate = "1/1/1997"
    strEndDate = "12/31/1997"
    strFile = CurrentProject.Path & "Orders.rst"

    'Specify the OLE DB provider and open the connection
    With cnnOrders
        .Provider = "SQLOLEDB.1"
        On Error Resume Next
        .Open "Data Source=(local);" & _
            "UID=sa;PWD=;Initial Catalog=NorthwindCS"
        If Err.Number Then
            .Open "Data Source=(local);" & _
                "Integrated Security=SSPI;Initial Catalog=NorthwindCS"
        End if
        On Error GoTo 0
    End With

    With cmmOrders
        'Create and append the BeginningDate parameter
        Set prmBegDate = .CreateParameter("BegDate", adChar, _
            adParamInput, Len(strBegDate), strBegDate)
        .Parameters.Append prmBegDate
        'Create and append the endingDate parameter
        Set prmEndDate = .CreateParameter("EndDate", adChar, _
            adParamInput, Len(strEndDate), strEndDate)
        .Parameters.Append prmEndDate
```

```
Set .ActiveConnection = cnnOrders
'Specify a stored procedure
.CommandType = adCmdStoredProc
'Brackets must surround stored procedure names with spaces
.CommandText = "[Sales By Year]"
'Receive the Recordset
Set rstOrders = .Execute 'returns a "firehose" Recordset
End With

With rstOrders
'Save (persist) the forward-only Recordset to a file
On Error Resume Next
'Delete the file, if it exists
Kill strFile
On Error GoTo 0
.Save strFile
.Close
.Open strFile, "Provider=MSPersist", , , adCmdFile
End With

'Assign rstOrders to the Recordset of the form
Set Me.Recordset = rstOrders

Me.txtShippedDate.ControlSource = "ShippedDate"
Me.txtOrderID.ControlSource = "OrderID"
Me.txtSubtotal.ControlSource = "Subtotal"
Me.txtYear.ControlSource = "Year"
End Sub
```

## CAUTION

When used in ADO code, you must enclose names of stored procedures and views having spaces with square brackets. Including spaces in database object names, especially in client/server environments, isn't a recommended practice. Microsoft developers insist on adding spaces in names of views and stored procedures, perhaps because SQL Server 2000 supports this dubious feature. Use underscores to make object names more readable if necessary.

## NOTE

The code of Listing 30.2 uses an ADO 2.5+ feature, persisted (saved) Recordset objects. Stored procedures return forward-only ("firehose") Recordset objects, which you can't assign to the Recordset property of a form. To create a Recordset with a cursor acceptable to Access forms, you must persist the Recordset as a file and then close and reopen the Recordset with the MSPersist OLE DB provider as the ActiveConnection property value. The "Recordset Methods" section, later in the chapter, provides the complete syntax for the Save and Open methods of the Recordset object.



Figure 30.18 shows the result of executing the code of Listing 30.2. The frmParams form that contains the code is included in the ADOTest.mdb and ADOTest.adp files described earlier in the chapter. The AddOrders.adp project, described in the “Exploring the AddOrders.adp Sample Project” section near the end of the chapter, also includes code for setting stored procedure parameter values.

**Figure 30.18**

This Datasheet view of the read-only Recordset returned by the Sales By Year stored procedure displays the value of each order received in 1997.

Ship Date	Order ID	Subtotal	Year
1/16/1997	10390	\$1,313.82	1997
1/1/1997	10392	\$1,440.00	1997
1/3/1997	10393	\$2,566.95	1997
1/3/1997	10394	\$442.00	1997
1/3/1997	10395	\$2,122.92	1997
1/6/1997	10396	\$1,903.80	1997
1/2/1997	10397	\$716.72	1997
1/9/1997	10398	\$2,505.60	1997
1/8/1997	10399	\$1,765.60	1997
1/16/1997	10400	\$3,063.00	1997
1/10/1997	10401	\$3,868.60	1997
1/10/1997	10402	\$2,713.50	1997
1/9/1997	10403	\$855.02	1997
1/8/1997	10404	\$1,591.25	1997
1/22/1997	10405	\$400.00	1997
1/13/1997	10406	\$1,830.78	1997
1/30/1997	10407	\$1,194.00	1997

## UNDERSTANDING THE ADODB.Recordset OBJECT

Creating, viewing, and updating Recordset objects is the ultimate objective of most Access database front ends. Opening an independent ADODB.Recordset object offers a myriad of cursor, locking, and other options. You must explicitly open a Recordset with a scrollable cursor if you want to use code to create the Recordset for assignment to the Form.Recordset property. Unlike Jet and ODBCDirect Recordset objects, ADODB.Recordset objects expose a number of events that are especially useful for validating Recordset updates.

### Recordset PROPERTIES

Microsoft attempted to make ADODB.Recordset objects backward compatible with DAO.Recordset objects to minimize the amount of code you must change to migrate existing applications from DAO to ADO. Unfortunately, the attempt at backward compatibility for code-intensive database applications didn't fully succeed. You must make substantial changes in DAO code to accommodate ADO's updated Recordset object. Thus, most Access developers choose ADO for new Access front-end applications and stick with DAO when maintaining existing Jet projects.

Table 30.15 lists the names and descriptions of the standard property set of ADODB.Recordset objects. ADODB.Recordset objects have substantially fewer properties than DAO.Recordset objects have. The standard properties of ADODB.Recordset objects are those that are supported by the most common OLE DB data providers for relational databases.

TABLE 30.15 PROPERTY NAMES AND DESCRIPTIONS FOR ADODB.Recordset OBJECTS

Property Name	Description
AbsolutePage	A <b>Long</b> read/write value that sets or returns the number of the page in which the current record is located or one of the constant values of <code>PositionEnum</code> (see Table 30.16). You must set the <code>PageSize</code> property value before getting or setting the value of <code>AbsolutePage</code> . <code>AbsolutePage</code> is 1 based; if the current record is in the first page, <code>AbsolutePage</code> returns 1. Setting the value of <code>AbsolutePage</code> causes the current record to be set to the first record of the specified page.
AbsolutePosition	A <b>Long</b> read/write value (1 based) that sets or returns the position of the current record. The maximum value of <code>AbsolutePosition</code> is the value of the <code>RecordCount</code> property.
ActiveCommand	A <b>VARIANT</b> read-only value specifying the name of a previously opened <code>Command</code> object with which the <code>Recordset</code> is associated.
ActiveConnection	A pointer to a previously opened <code>Connection</code> object with which the <code>Recordset</code> is associated or a fully qualified <code>ConnectionString</code> value.
BOF	A <b>Boolean</b> read-only value that, when <code>True</code> , indicates that the record pointer is positioned before the first row of the <code>Recordset</code> and there is no current record.
Bookmark	A <b>VARIANT</b> read/write value that returns a reference to a specific record or uses a <code>Bookmark</code> value to set the record pointer to a specific record.
CacheSize	A <b>Long</b> read/write value that specifies the number of records stored in local (cache) memory. The minimum (default) value is 1. Increasing the value of <code>CacheSize</code> minimizes round trips to the server to obtain additional rows when scrolling through <code>Recordset</code> objects.
CursorLocation	A <b>Long</b> read/write value that specifies the location of a scrollable cursor, subject to the availability of the specified <code>CursorType</code> on the client or server (see Table 30.17). The default is to use a cursor supplied by the OLE DB data source (called a <i>server-side</i> cursor).
CursorType	A <b>Long</b> read/write value that specifies the type of <code>Recordset</code> cursor (see Table 30.18). The default is a forward-only (fire hose) cursor.
DataMember	Returns a pointer to an associated <code>Command</code> object created by Visual Basic's Data Environment Designer.
DataSource	Returns a pointer to an associated <code>Connection</code> object.
EditMode	A <b>Long</b> read-only value that returns the status of editing of the current record (see Table 30.19).

*continues*

TABLE 30.15 CONTINUED

Property Name	Description
EOF	A <b>Boolean</b> read-only value that, when True, indicates that the record pointer is beyond the last row of the <code>Recordset</code> and there is no current record.
Fields	A pointer to the <code>Fields</code> collection of <code>Field</code> objects of the <code>Recordset</code> .
Filter	A <b>Variant</b> read/write value that can be a criteria string (a valid SQL WHERE clause without the WHERE reserved word), an array of <code>Bookmark</code> values specifying a particular set of records, or a constant value from <code>FilterGroupEnum</code> (see Table 30.20).
Index	A <b>String</b> read/write value that sets or returns the name of an existing index on the base table of the <code>Recordset</code> . The <code>Recordset</code> must be closed to set the <code>Index</code> value to the name of an index. The <code>Index</code> property is used primarily in conjunction with the <code>Recordset.Seek</code> method.
LockType	A <b>Long</b> read/write value that specifies the record-locking method employed when opening the <code>Recordset</code> (see Table 30.21). The default is read-only, corresponding to the read-only characteristic of forward-only cursors.
MarshalOptions	A <b>Long</b> read/write value that specifies which set of records is returned to the server after client-side modification. The <code>MarshalOptions</code> property applies only to the lightweight <code>ADOR.Recordset</code> object, a member of RDS.
MaxRecords	A <b>Long</b> read/write value that specifies the maximum number of records to be returned by a <code>SELECT</code> query or stored procedure. The default value is 0, all records.
PageCount	A <b>Long</b> read-only value that returns the number of pages in a <code>Recordset</code> . You must set the <code>PageSize</code> value to cause <code>PageCount</code> to return a meaningful value. If the <code>Recordset</code> doesn't support the <code>PageCount</code> property, the value is -1.
PageSize	A <b>Long</b> read/write value that sets or returns the number of records in a logical page. You use logical pages to break large <code>Recordsets</code> into easily manageable chunks. <code>PageSize</code> isn't related to the size of table pages used for locking in Jet (2KB) or SQL Server (2KB in version 6.5 and earlier, 8KB in version 7+) databases.

Property Name	Description
PersistFormat	A <b>Long</b> read/write value that sets or returns the format of Recordset files created by calling the Save method. The two constant values of PersistFormatEnum are adPersistADTG (the default format, Advanced Data TableGram or ADTG) and adPersistXML, which saves the Recordset as almost-readable XML. The XML schema, rowset, is a variation of the XML Data Reduced (XDR) schema, a Microsoft-only attribute-centric namespace that isn't compatible with Access's XSD (XML Schema) format.
Properties	A pointer to the Properties collection of provider-specific Property values of the Recordset.
RecordCount	A <b>Long</b> read-only value that returns the number of records in Recordset objects with scrollable cursors if the Recordset supports approximate positioning or Bookmarks. (See the Recordset.Supports method later in this chapter.) If not, you must apply the MoveLast method to obtain an accurate RecordCount value, which retrieves and counts all records. If a forward-only Recordset has one or more records, RecordCount returns -1 ( <b>True</b> ). An empty Recordset of any type returns 0 ( <b>False</b> ).
Sort	A <b>String</b> read/write value, consisting of a valid SQL ORDER BY clause without the ORDER BY reserved words, which specifies the sort order of the Recordset.
Source	A <b>String</b> read/write value that can be an SQL statement, a table name, a stored procedure name, or the name of an associated Command object. If you supply the name of a Command object, the Source property returns the value of the Command.CommandText property as text. Use the lngOptions argument of the Open method to specify the type of the value supplied to the Source property.
State	A <b>Long</b> read/write value representing one of the constant values of ObjectStateEnum (refer to Table 30.4).
Status	A <b>Long</b> read-only value that indicates the status of batch operations or other multiple-record (bulk) operations on the Recordset (see Table 30.22).
StayInSync	A <b>Boolean</b> read/write value, which, if set to <b>True</b> , updates references to child (chapter) rows when the parent row changes. StayInSync applies only to hierarchical Recordset objects.



The most obvious omission in the preceding table is the `DAO.Recordset.NoMatch` property value used to test whether applying one of the `DAO.Recordset.Find...` methods or the `DAO.Recordset.Seek` method succeeds. The new `ADODB.Recordset.Find` method, listed in the “Recordset Methods” section later in this chapter, substitutes for DAO’s `FindFirst`, `FindNext`, `FindPrevious`, and `FindLast` methods. The `Find` method uses the `EOF` property value for testing the existence of one or more records matching the `Find` criteria.

Another omission in the `ADODB.Recordset` object’s preceding property list is the `PercentPosition` property. The workaround, however, is easy:

```
rstName.AbsolutePosition = Int(intPercentPosition * rstName.RecordCount / 100)
```

Tables 30.16 through 30.22 enumerate the valid constant values for the `AbsolutePage`, `CursorLocation`, `CursorType`, `EditMode`, `Filter`, `LockType`, and `Status` properties. Default values appear first, if defined; the list of remaining enumeration members is ordered by frequency of use in Access applications.

**TABLE 30.16** CONSTANT VALUES FOR THE `AbsolutePage` PROPERTY

<code>AbsolutePageEnum</code>	Description
<code>adPosUnknown</code>	The data provider doesn’t support pages, the <code>Recordset</code> is empty, or the data provider can’t determine the page number.
<code>adPosBOF</code>	The record pointer is positioned at the beginning of the file. (The <code>BOF</code> property is <b>True</b> .)
<code>adPosEOF</code>	The record pointer is positioned at the end of the file. (The <code>EOF</code> property is <b>True</b> .)

**TABLE 30.17** CONSTANT VALUES FOR THE `CursorLocation` PROPERTY

<code>CursorLocationEnum</code>	Description
<code>adUseClient</code>	Use cursor(s) provided by a cursor library located on the client. The <code>ADOR.Recordset</code> (RDS) requires a client-side cursor.
<code>adUseServer</code>	Use cursor(s) supplied by the data source, usually (but not necessarily) located on a server (default value).

TABLE 30.18 CONSTANT VALUES FOR THE `CursorType` PROPERTY

<code>CursorLocationEnum</code>	Description
<code>adOpenForwardOnly</code>	Provides only unidirectional cursor movement and a read-only <code>Recordset</code> (default value).
<code>adOpenDynamic</code>	Provides a scrollable cursor that displays all changes, including new records, which other users make to the <code>Recordset</code> .
<code>adOpenKeyset</code>	Provides a scrollable cursor that hides only records added or deleted by other users; similar to a <code>DAO.Recordset</code> of the <code>dynaset</code> type.
<code>adOpenStatic</code>	Provides a scrollable cursor over a static copy of the <code>Recordset</code> . Similar to a <code>DAO.Recordset</code> of the <code>snapshot</code> type, but the <code>Recordset</code> is updatable.

TABLE 30.19 CONSTANT VALUES FOR THE `EditMode` PROPERTY

<code>EditModeEnum</code>	Description
<code>adEditNone</code>	No editing operation is in progress (default value).
<code>adEditAdd</code>	A tentative append record has been added, but not saved to the database table(s).
<code>adEditDelete</code>	The current record has been deleted.
<code>adEditInProgress</code>	Data in the current record has been modified, but not saved to the database table(s).

TABLE 30.20 CONSTANT VALUES FOR THE `Filter` PROPERTY

<code>FilterGroupEnum</code>	Description
<code>adFilterNone</code>	Removes an existing filter and exposes all records of the <code>Recordset</code> (equivalent to setting the <code>Filter</code> property to an empty string, the default value).
<code>adFilterAffectedRecords</code>	View only records affected by the last execution of the <code>CancelBatch</code> , <code>Delete</code> , <code>Resync</code> , or <code>UpdateBatch</code> method.
<code>adFilterFetchedRecords</code>	View only records in the current cache. The number of records is set by the <code>CacheSize</code> property.
<code>adFilterConflictingRecords</code>	View only records that failed to update during the last batch update operation.
<code>adFilterPendingRecords</code>	View only records that have been modified but not yet processed by the data source (for <code>Batch Update</code> mode only).

**TABLE 30.21** CONSTANT VALUES FOR THE LockType PROPERTY

LockTypeEnum	Description
adLockReadOnly	Specifies read-only access (default value).
adLockBatchOptimistic	Use Batch Update mode instead of the default Immediate Update mode.
adLockOptimistic	Use optimistic locking (lock the record or page only during the table update process).
adLockPessimistic	Use pessimistic locking (lock the record or page during editing and the updated process).
adLockUnspecified	No lock type specified. (Use this constant only for Recordset clones.)

**TABLE 30.22** CONSTANT VALUES FOR THE Status PROPERTY (APPLIES TO BATCH OR BULK Recordset OPERATIONS ONLY)

RecordStatusEnum	Description of Record Status
adRecOK	Updated successfully
adRecNew	Added successfully
adRecModified	Modified successfully
adRecDeleted	Deleted successfully
adRecUnmodified	Not modified
adRecInvalid	Not saved; the Bookmark property is invalid
adRecMultipleChanges	Not saved; saving would affect other records
adRecPendingChanges	Not saved; the record refers to a pending insert operation)
adRecCanceled	Not saved; the operation was canceled
adRecCantRelease	Not saved; existing record locks prevented saving
adRecConcurrencyViolation	Not saved; an optimistic concurrency locking problem occurred
adRecIntegrityViolation	Not saved; the operation would violate integrity constraints
adRecMaxChangesExceeded	Not saved; an excessive number of pending changes exist
adRecObjectOpen	Not saved; a conflict with an open storage object occurred
adRecOutOfMemory	Not saved; the machine is out of memory
adRecPermissionDenied	Not saved; the user doesn't have required permissions
adRecSchemaViolation	Not saved; the record structure doesn't match the database schema
adRecDBDeleted	Not saved or deleted; the record was previously deleted

## Fields COLLECTION AND Field OBJECTS

Like DAO's `Fields` collection, ADO's dependent `Fields` collection is a property of the `Recordset` object, making the columns of the `Recordset` accessible to VBA code and bound controls. The `Fields` collection has one property, `Count`, and only two methods, `Item` and `Refresh`. You can't append new `Field` objects to the `Fields` collection, unless you're creating a persisted `Recordset` from scratch or you use ADOX's `ALTER TABLE DDL` command to add a new field.

All but one (`Value`) of the property values of `Field` objects are read-only, because the values of the `Field` properties are derived from the database schema. The `Value` property is read-only in forward-only `Recordsets` and `Recordsets` opened with read-only locking. Table 30.23 gives the names and descriptions of the properties of the `Field` object.

**TABLE 30.23** PROPERTY NAMES AND DESCRIPTIONS OF THE `Field` OBJECT

<b>Field Property</b>	<b>Description</b>
<code>ActualSize</code>	A <b>Long</b> read-only value representing the length of the <code>Field</code> 's value by character count.
<code>Attributes</code>	A <b>Long</b> read-only value that represents the sum of the constants (flags) included in <code>FieldAttributeEnum</code> (see Table 30.24).
<code>DefinedSize</code>	A <b>Long</b> read-only value specifying the maximum length of the <code>Field</code> 's value by character count.
<code>Name</code>	A <b>String</b> read-only value that returns the field (column) name.
<code>NumericScale</code>	A <b>Byte</b> read-only value specifying the number of decimal places for numeric values.
<code>OriginalValue</code>	A <b>VARIANT</b> read-only value that represents the <code>Value</code> property of the field before applying the <code>Update</code> method to the <code>Recordset</code> . (The <code>CancelUpdate</code> method uses <code>OriginalValue</code> to replace a changed <code>Value</code> property.)
<code>Precision</code>	A <b>Byte</b> read-only value specifying the total number of digits (including decimal digits) for numeric values.
<code>Properties</code>	A collection of provider-specific <code>Property</code> objects. SQL Server 2000's extended properties are an example <code>Properties</code> collection members for the SQL Server OLE DB provider.
<code>Status</code>	An undocumented <b>Long</b> read-only value.
<code>Type</code>	A <b>Long</b> read-only value specifying the data type of the field. Refer to Table 30.14 for <code>Type</code> constant values.
<code>UnderlyingValue</code>	A <b>VARIANT</b> read-only value representing the current value of the field in the database table(s). You can compare the values of <code>OriginalValue</code> and <code>UnderlyingValue</code> to determine whether a persistent change has been made to the database, perhaps by another user.
<code>Value</code>	A <b>VARIANT</b> read/write value of a subtype appropriate to the value of the <code>Type</code> property for the field. If the <code>Recordset</code> isn't updatable, the <code>Value</code> property is read-only.

Value is the default property of the `Field` object, but it's a good programming practice to set and return field values by explicit use of the `Value` property name in VBA code. In most cases, using `varName = rstName.Fields(n).Value` instead of `varName = rstName.Fields(n)` results in a slight performance improvement.

**TABLE 30.24** CONSTANT VALUES AND DESCRIPTIONS FOR THE ATTRIBUTES PROPERTY OF THE `Field` OBJECT

<b>FieldAttributeEnum</b>	<b>Description</b>
<code>adFldCacheDeferred</code>	The provider caches field values. Multiple reads are made on the cached value, not the database table.
<code>adFldDefaultStream</code>	The field contains a stream of bytes. For example, the field might contain the HTML stream from a Web page specified by a field whose <code>adFldIsRowURL</code> attribute is <b>True</b> .
<code>adFldFixed</code>	The field contains fixed-length data with the length determined by the data type or field specification.
<code>adFldIsChapter</code>	The field is a member of a chaptered recordset and contains a child recordset of this field.
<code>adFldIsCollection</code>	The field contains a reference to a collection of resources, rather than a single resource.
<code>adFldIsNullable</code>	The field accepts <b>Null</b> values.
<code>adFldIsRowURL</code>	The field contains a URL for a resource such as a Web page.
<code>adFldKeyColumn</code>	The field is the primary key field of a table.
<code>adFldLong</code>	The field has a long binary data type, which permits the use of the <code>AppendChunk</code> and <code>GetChunk</code> methods.
<code>adFldMayBeNull</code>	The field can return <b>Null</b> values.
<code>adFldMayDefer</code>	The field is deferrable, meaning that <code>Values</code> are retrieved from the data source only when explicitly requested.
<code>adFldNegativeScale</code>	The field contains data from a column that supports negative <code>Scale</code> values.
<code>adFldRowID</code>	The field is a row identifier (typically an <code>identity</code> , <code>AutoIncrement</code> , or <code>GUID</code> data type).
<code>adFldRowVersion</code>	The field contains a timestamp or similar value for determining the time of the last update.
<code>adFldUpdatable</code>	The field is read/write (updatable).
<code>adFldUnknownUpdatable</code>	The data provider can't determine whether the field is updatable. Your only recourse is to attempt an update and trap the error that occurs if the field isn't updatable.

The `Field` object has two methods, `AppendChunk` and `GetChunk`, which are applicable only to fields of various long binary data types, indicated by an `adFldLong` flag in the `Attributes` property of the field. The `AppendChunk` method is discussed in the “Parameter Object” section earlier in this chapter. The syntax for the `AppendChunk` method call, which writes **Variant** data to a long binary field (`fldName`), is

```
fldName.AppendChunk varData
```

#### NOTE

ADO 2.x doesn't support the Access OLE Object field data type, which adds a proprietary object wrapper around the data (such as a bitmap) to identify the OLE server that created the object (for bitmaps, usually Windows Paint).

30

The `GetChunk` method enables you to read long binary data in blocks of the size you specify. Following is the syntax for the `GetChunk` method:

```
varName = fldName.GetChunk(lngSize)
```

A common practice is to place `AppendChunk` and `GetChunk` method calls within **Do Until...Loop** structures to break up the long binary value into chunks of manageable size. In the case of the **GetChunk** method, if you set the value of `lngSize` to less than the value of the field's `ActualSize` property, the first `GetChunk` call retrieves `lngSize` bytes. Successive `GetChunk` calls retrieve `lngSize` bytes beginning at the next byte after the end of the preceding call. If the remaining number of bytes is less than `lngSize`, only the remaining bytes appear in `varName`. After you retrieve the field's bytes, or if the field is empty, `GetChunk` returns **Null**.

#### NOTE

Changing the position of the record pointer of the field's `Recordset` resets `GetChunk`'s byte pointer. Accessing a different `Recordset` and moving its record pointer doesn't affect the other `Recordset`'s `GetChunk` record pointer.

## Recordset METHODS

`ADODB.Recordset` methods are an amalgam of the `DAO.Recordset` and `rdoResultset` methods. Table 30.25 gives the names, descriptions, and calling syntax for `Recordset` methods. OLE DB data providers aren't required to support all the methods of the `Recordset` object. If you don't know which methods the data provider supports, you must use the `Supports` method with the appropriate constant from `CursorOptionEnum`, listed in Table 30.28 later in this chapter, to test for support of methods that are provider dependent. Provider-dependent methods are indicated by an asterisk after the method name in Table 30.25.

TABLE 30.25 NAMES AND DESCRIPTIONS OF METHODS OF THE Recordset OBJECT

Method Name	Description and Calling Syntax
AddNew	Adds a new record to an updatable Recordset. The calling syntax is <code>rstName.AddNew [{varField avarFields}, {varValue avarValues}]</code> , where <code>varField</code> is a single field name, <code>avarFields</code> is an array of field names, <code>varValue</code> is single value, and <code>avarValues</code> is an array of values for the columns defined by the members of <code>avarFields</code> . Calling the <code>Update</code> method adds the new record to the database table(s). If you add a new record to a Recordset having a primary-key field that isn't the first field of the Recordset, you must supply the name and value of the primary-key field in the <code>AddNew</code> statement.
Cancel	Cancels execution of an asynchronous query and terminates creation of multiple Recordsets from stored procedures or compound SQL statements. The calling syntax is <code>rstName.Cancel</code> .
CancelBatch	Cancels a pending batch update operation on a Recordset whose <code>LockEdits</code> property value is <code>adBatchOptimistic</code> . The calling syntax is <code>rstName.CancelBatch [lngAffectRecords]</code> . The optional <code>lngAffectRecords</code> argument is one of the constants of <code>AffectEnum</code> (see Table 30.26).
CancelUpdate	Cancels a pending change to the table(s) underlying the Recordset before applying the <code>Update</code> method. The calling syntax is <code>rstName.CancelUpdate</code> .
Clone	Creates a duplicate Recordset object with an independent record pointer. The calling syntax is <code>Set rstDupe = rstName.Clone()</code> .
Close	Closes a Recordset object, allowing reuse of the Recordset variable by setting new Recordset property values and applying the <code>Open</code> method. The calling syntax is <code>rstName.Close</code> .
CompareBookmarks	Returns the relative value of two bookmarks in the same Recordset or a Recordset and its clone. The calling syntax is <code>lngResult = rstName.CompareBookmarks(varBookmark1, varBookmark2)</code> .
Delete	Deletes the current record immediately from the Recordset and the underlying tables, unless the <code>LockEdits</code> property value of the Recordset is set to <code>adLockBatchOptimistic</code> . The calling syntax is <code>rstName.Delete</code> .



Method Name	Description and Calling Syntax
Find	Searches for a record based on criteria you supply. The calling syntax is <code>rstName.Find strCriteria[, lngSkipRecords, lngSearchDirection[, lngStart]]</code> , where <code>strCriteria</code> is a valid SQL WHERE clause without the WHERE keyword, the optional <code>lngSkipRecords</code> value is the number of records to skip before applying Find, <code>lngSearchDirection</code> specifies the search direction ( <code>adSearchForward</code> , the default, or <code>adSearchBackward</code> ), and the optional <code>varStart</code> value specifies the Bookmark value of the record at which to start the search or one of the members of <code>BookmarkEnum</code> (see Table 30.27). If Find succeeds, the EOF property returns <b>False</b> ; otherwise, EOF returns <b>True</b> .
GetRows	Returns a two-dimensional (row, column) <b>Variant</b> array of records. The calling syntax is <code>avarName = rstName.GetRows(lngRows[, varStart[, {strFieldName!lngFieldIndex!avarFieldNames!avarFieldIndexes}]])</code> , where <code>lngRows</code> is the number of rows to return, <code>varStart</code> specifies a Bookmark value of the record at which to start the search or one of the members of <code>BookmarkEnum</code> (see Table 30.27), and the third optional argument is the name or index of a single column, or a <b>Variant</b> array of column names or indexes. If you don't specify a value of the third argument, GetRows returns all columns of the Recordset.
GetString	By default, returns a tab-separated <b>String</b> value for a specified number of records, with records separated by return codes. The calling syntax is <code>strClip = rstName.GetString(lngRows[, strColumnDelimiter[, strRowDelimiter, [strNullExpr]])</code> , where <code>lngRows</code> is the number of rows to return, <code>strColumnDelimiter</code> is an optional column-separation character ( <code>vbTab</code> is the default), <code>strRowDelimiter</code> is an optional row-separation character ( <code>vbCR</code> is the default), and <code>strNullExpr</code> is an optional value to substitute when encountering <b>Null</b> values (an empty string, "", is the default value).
Move	Moves the record pointer from the current record. The calling syntax is <code>rstName.Move lngNumRecords[, varStart]</code> , where <code>lngNumRecords</code> is the number of records by which to move the record pointer and the optional <code>varStart</code> value specifies the Bookmark of the record at which to start the search or one of the members of <code>BookmarkEnum</code> (see Table 30.27).

*continues*

TABLE 30.25 CONTINUED

Method Name	Description and Calling Syntax
MoveFirst	Moves the record pointer to the first record. The calling syntax is <code>rstName.MoveFirst</code> .
MoveLast	Moves the record pointer to the last record. The calling syntax is <code>rstName.MoveLast</code> .
MoveNext	Moves the record pointer to the next record. The calling syntax is <code>rstName.MoveNext</code> . The <code>MoveNext</code> method is the only <code>Move...</code> method that you can apply to a forward-only <code>Recordset</code> .
MovePrevious	Moves the record pointer to the previous record. The calling syntax is <code>rstName.MovePrevious</code> .
NextRecordset	Returns additional <code>Recordset</code> objects generated by a compound Jet SQL statement, such as <code>SELECT * FROM Orders</code> ; <code>SELECT * FROM Customers</code> , or a T-SQL stored procedure that returns multiple <code>Recordsets</code> . The calling syntax is <code>rstNext = rstName.NextRecordset [(lngRecordsAffected)]</code> , where <code>lngRecordsAffected</code> is an optional return value that specifies the number of records in <code>rstNext</code> , if <code>SET NOCOUNT ON</code> isn't included in the SQL statement or stored procedure code. If no additional <code>Recordset</code> exists, <code>rstNext</code> is set to <b>Nothing</b> .
Open	Opens a <code>Recordset</code> on an active <code>Command</code> or <code>Connection</code> object. The calling syntax is <code>rstName.Open [varSource[, varActiveConnection[, lngCursorType[, lngLockType[, lngOptions]]]]</code> . The <code>Open</code> arguments are optional if you set the equivalent <code>Recordset</code> property values, which is the practice recommended in this book. For valid values, refer to the <code>Source</code> , <code>ActiveConnection</code> , <code>CursorType</code> , and <code>LockType</code> properties in Table 30.15 earlier in this chapter and to the <code>CommandTypeEnum</code> values listed in Table 30.7 earlier in this chapter for the <code>lngOptions</code> property.
Requery	Refreshes the content of the <code>Recordset</code> from the underlying table(s), the equivalent of calling <code>Close</code> and then <code>Open</code> . <code>Requery</code> is a very resource-intensive operation. The calling syntax is <code>rstName.Requery</code> .
Resync	Refreshes a specified subset of the <code>Recordset</code> from the underlying table(s). The calling syntax is <code>rstName.Resync [lngAffectRecords]</code> , where <code>lngAffectRecords</code> is one of the members of <code>AffectEnum</code> (see Table 30.26). If you select <code>adAffectCurrent</code> or <code>adAffectGroup</code> as the value of <code>lngAffectRecords</code> , you reduce the required resources in comparison with <code>adAffectAll</code> (the default).

Method Name	Description and Calling Syntax
Save	Creates a file containing a persistent copy of the Recordset. The calling syntax is <code>rstName.Save strFileName</code> , where <code>strFileName</code> is the path to and the name of the file. You open a Recordset from a file with a <code>rstName.Open strFileName, Options:=adCmdFile</code> statement. This book uses <code>.rst</code> as the extension for persistent Recordsets in the ADTG format and <code>.xml</code> for XML formats.
Seek	Performs a high-speed search on the field whose index name is specified as the value of the <code>Recordset.Index</code> property. The calling syntax is <code>rstName.Seek avarKeyValues[, lngOption]</code> , where <code>avarKeyValues</code> is a <b>Variant</b> array of search values for each field of the index. The optional <code>lngOption</code> argument is one of the members of the <code>SeekEnum</code> (see Table 30.29) constant enumeration; the default value is <code>adSeekFirstEQ</code> (find the first equal value). You can't specify <code>adUseClient</code> as the <code>CursorLocation</code> property value when applying the <code>Seek</code> method; <code>Seek</code> requires a server-side ( <code>adUseServer</code> ) cursor.
Supports	Returns <b>True</b> if the Recordset's data provider supports a specified cursor-dependent method; otherwise, <code>Supports</code> returns <b>False</b> . The calling syntax is <code>blnSupported = rstName.Supports(lngCursorOptions)</code> . Table 30.28 lists the names and descriptions of the <code>CursorOptionEnum</code> values.
Update	Applies the result of modifications to the Recordset to the underlying table(s) of the data source. For batch operations, <code>Update</code> applies the modifications only to the local (cached) Recordset. The calling syntax is <code>rstName.Update</code> .
UpdateBatch	Applies the result of all modifications made to a batch-type Recordset ( <code>LockType</code> property set to <code>adBatchOptimistic</code> , <code>CursorType</code> property set to <code>adOpenKeyset</code> or <code>adOpenStatic</code> , and <code>CursorLocation</code> property set to <code>adUseClient</code> ) to the underlying table(s) of the data source. The calling syntax is <code>rstName.UpdateBatch [lngAffectRecords]</code> , where <code>lngAffectRecords</code> is a member of <code>AffectEnum</code> (see Table 30.26).

The “Code to Pass Parameter Values to a Stored Procedure” section, earlier in the chapter, illustrates use of the `Save` and `Open` methods with persisted Recordsets of the ADTG type.

## TIP

The `Edit` method of `DAO.Recordset` objects is missing from Table 30.25. To change the value of one or more fields of the current record of an `ADODB.Recordset` object, execute `rstName.Fields(n).Value = varValue` for each field whose value you want to change and then execute `rstName.Update`. `ADODB.Recordset` objects don't support the `Edit` method.

To improve the performance of `Recordset` objects opened on `Connection` objects, set the required property values of the `Recordset` object and then use a named argument to specify the `intOptions` value of the `Open` method, as in `rstName.Open Options:=adCmdText`. This syntax is easier to read and less prone to error than the alternative, `rstName.Open , , , , adCmdText`.

**TABLE 30.26** NAMES AND DESCRIPTIONS OF CONSTANTS FOR THE `CancelBatch` METHOD'S `lngAffectRecords` ARGUMENT

<b>AffectEnum</b>	<b>Description</b>
<code>adAffectAll</code>	Include all records in the <code>Recordset</code> object, including any records hidden by the <code>Filter</code> property value (the default)
<code>adAffectAllChapters</code>	Include all chapter fields in a chaptered recordset, including any records hidden by the <code>Filter</code> property value.
<code>adAffectCurrent</code>	Include only the current record
<code>adAffectGroup</code>	Include only those records that meet the current <code>Filter</code> criteria

**TABLE 30.27** NAMES AND DESCRIPTIONS OF `Bookmark` CONSTANTS FOR THE `Find` METHOD'S `varStart` ARGUMENT

<b>BookmarkEnum</b>	<b>Description</b>
<code>adBookmarkCurrent</code>	Start at the current record (the default value)
<code>adBookmarkFirst</code>	Start at the first record
<code>adBookmarkLast</code>	Start at the last record

**TABLE 30.28** NAMES AND DESCRIPTIONS OF CONSTANTS FOR THE `Supports` METHOD

<b>CursorOptionEnum</b>	<b>Permits</b>
<code>adAddNew</code>	Applying the <code>AddNew</code> method
<code>adApproxPosition</code>	Setting and getting <code>AbsolutePosition</code> and <code>AbsolutePage</code> property values
<code>adBookmark</code>	Setting and getting the <code>Bookmark</code> property value
<code>adDelete</code>	Applying the <code>Delete</code> method

CursorOptionEnum	Permits
adFind	Applying the Find method
adHoldRecords	Retrieving additional records or changing the retrieval record pointer position without committing pending changes
adIndex	Use of the Index property
adMovePrevious	Applying the GetRows, Move, MoveFirst, and MovePrevious methods (indicates a bidirectional scrollable cursor)
adNotify	Use of Recordset events
adResync	Applying the Resync method
adSeek	Applying the Seek method
adUpdate	Applying the Update method
adUpdateBatch	Applying the UpdateBatch and CancelBatch methods

Table 30.29 lists the SeekEnum constants for the optional lngSeekOptions argument of the Seek method. Unfortunately, the syntax for the ADODB.Recordset.Seek method isn't even close to being backward-compatible with the DAO.Recordset.Seek method.

**TABLE 30.29** NAMES AND DESCRIPTIONS OF CONSTANTS FOR THE Seek METHOD'S lngSeekOptions ARGUMENT

SeekEnum	Finds
adSeekFirstEQ	The first equal value (the default value)
adSeekAfterEQ	The first equal value or the next record after which a match would have occurred (logical equivalent of >=)
adSeekAfter	The first record after which an equal match would have occurred (logical equivalent of >)
adSeekBeforeEQ	The first equal value or the previous record before which a match would have occurred (logical equivalent of <=)
adSeekBefore	The first record previous to where an equal match would have occurred (logical equivalent of <)
adSeekLastEQ	The last record having an equal value

#### TIP

Use the Find method for searches unless you are working with a table having an extremely large number of records. Find takes advantage of index(es), if present, but Find's search algorithm isn't quite as efficient as Seek's. You'll probably encounter the threshold for considering substituting Seek for Find in the range of 500,000 to 1,000,000 records. Tests on a large version the Oakmont.mdb Jet and Oakmont SQL Server Students table (50,000) rows show imperceptible performance differences between Seek and Find operations.

## Recordset EVENTS

Recordset events are new to users of DAO. Table 30.30 names the Recordset events and gives the condition under which the event fires.

**TABLE 30.30 NAMES AND OCCURRENCE OF RECORDSET EVENTS**

Event Name	When Fired
EndOfRecordset	When the record pointer attempts to move beyond the last record
FetchComplete	When all records have been retrieved asynchronously
FetchProgress	During asynchronous retrieval, periodically reports the number of records returned
FieldChangeComplete	After a change to the value of a field
MoveComplete	After execution of the Move or Move... methods
RecordChangeComplete	After an edit to a single record
RecordsetChangeComplete	After cached changes are applied to the underlying tables
WillChangeField	Before a change to a field value
WillChangeRecord	Before an edit to a single record
WillChangeRecordset	Before cached changes are applied to the underlying tables
WillMove	Before execution of the Move or Move... methods

## TAKING ADVANTAGE OF DISCONNECTED Recordsets

If you set the value of the Recordset's `LockEdits` property to `adBatchOptimistic` and the `CursorType` property to `adKeyset` or `adStatic`, you create a batch-type Recordset object that you can disconnect from the data source. You can then edit the Recordset object offline with a client-side cursor, reopen the Connection object, and send the updates to the data source over the new connection. A Recordset without an active connection is called a *disconnected Recordset*. The advantage of a disconnected Recordset is that you eliminate the need for an active server connection during extended editing sessions. Batch updates solve the Access front-end scalability issues mentioned at the beginning of the chapter.

### NOTE



Unfortunately, you can't assign a disconnected Recordset to the `Recordset` property of a form or subform and take advantage of batch updates. Bound forms require an active connection to the database. You must write VBA code to handle updating, adding, and deleting records.

To learn more about updatability issues with disconnected Recordsets and the Client Data Manager (CDM) added by Access 2002, open Microsoft Knowledge Base article Q301987, "Using ADO in Microsoft Access 2002" and download the white paper.

Batch updates with disconnected Recordsets are stateless and resemble the interaction of Web browsers and servers when displaying conventional Web pages. The term *stateless* means that the current interaction between the client application and the server isn't dependent on the outcome of previous interactions. For example, you can make local updates to a disconnected Recordset, go to lunch, make additional updates as needed, and then send the entire batch to the server. A properly designed batch update application lets you close the application or shut down the client computer, and then resume the updating process when you restart the application.

**TIP**

Disconnected Recordsets minimize the effect of MSDE “five-user tuning” on the performance of Access online transaction processing (OLTP) applications. Batch updates execute very quickly, so most user connections remain open for a second or less.

Transaction processing with stored procedures or T-SQL statements that incorporate BEGIN TRANS...COMMIT TRANS...ROLLBACK TRANS statements are the better choice for OLTP operations on multiple tables, such as order-entry systems. It's possible for batch updates to succeed partially, which might result in a missing line item. You can use the Errors collection to analyze and potentially correct such problems, but doing so requires high-level VBA coding skills.

30

## THE BASICS OF DISCONNECTING AND RECONNECTING RECORDSETS

Following is an example of VBA pseudocode that creates and operates on a disconnected Recordset and then uses the UpdateBatch method to persist the changes in the data source:

```
Set rstName = New ADODB.Recordset
With rstName
    .ActiveConnection = cnnName
    .CursorType = adKeyset
    .CursorLocation = adUseClient
    .LockEdits = adBatchOptimistic
    .Open "SELECT * FROM TableName WHERE Criteria", Options:=adCmdText
    Set .ActiveConnection = Nothing 'Disconnect the Recordset
    'Close the connection to the server, if desired
    'Edit the field values of multiple records here
    'You also can append and delete records
    'Reopen the server connection, if closed
    Set .ActiveConnection = cnnName
    .UpdateBatch 'Send all changes to the data source
End With
rstName.Close
```

If calling the UpdateBatch method causes conflicts with other users' modifications to the underlying table(s), you receive a trappable error and the Errors collection contains Error object(s) that identify the conflict(s). Unlike transactions, which require all attempted modifications to succeed or all to be rolled back, Recordset batch modifications that don't cause conflicts are made permanent in the data source.

## AN EXAMPLE BATCH UPDATE APPLICATION



The frmBatchUpdate form of the ADOTest.mdb application and ADOTest.adp project demonstrates the effectiveness of batch updates with MSDE. For example, you can edit data, persist the edited disconnected Recordset as an ADTG or XML file, close the form (or Access), and then reopen the form and submit the changes to the server. A subform, sbfBatchUpdate, which is similar to the frmADO\_Jet and frmADO\_MSDE forms you created early in the chapter, displays the original and updated data. The subform is read-only; VBA code simulates user updates to the data. The example also demonstrates how to use VBA code to display the XML representation of a Recordset object in Internet Explorer (IE) 5+.

To give frmBatchUpdate a trial run, do this:



1. If you haven't installed the entire set of sample applications, copy ADOTest.mdb, ADOTest.adp, or both from the \Seua11\Chaptr30 folder of the accompanying CD-ROM to a ... \Program Files\Seua11\Chapter30 folder.
2. Verify that your local MSDE instance is running, and then open frmBatchUpdate, which connects to the NorthwindCS database, opens a Recordset, saves it in a local file named Batch.rst, and closes the connection. The subform displays the first 10 rows of seven fields of the Customers table by opening a Recordset from the local Batch.rst file (see Figure 30.19).

**Figure 30.19**

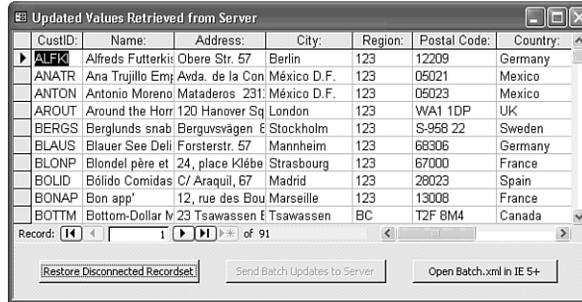
Opening frmBatchUpdate displays a disconnected Recordset opened on the Customers table for batch updates and saves the initial Recordset as Batch.xml.

CustID	Name	Address	City	Region	Postal Code	Country
ALFKI	Alfreds Futterk...	Obere Str. 57	Berlin		12209	Germany
ANATR	Ana Trujillo Emj	Avda. de la Con	México D.F.		05021	Mexico
ANTON	Antonio Moreno	Mataaderos 231	México D.F.		05023	Mexico
AROUT	Around the Horri	120 Hanover Sq	London		WA1 1DP	UK
BERGS	Berglunds snab	Berguvsvägen E	Stockholm		S-958 22	Sweden
BLAUS	Blauer See Deli	Forsterstr. 57	Mannheim		68306	Germany
BLONP	Blondel père et	24, place Klébe	Strasbourg		67000	France
BOLID	Bólido Comidas	C/ Araquil, 67	Madrid		28023	Spain
BONAP	Bon app'	12, rue des Bou	Marseille		13008	France
BOTTM	Bottom-Dollar	123 Tsawassen E	Tsawassen	BC	T2F 8M4	Canada

3. Click the Update Disconnected Recordset button to replace NULL values in the Region cells with 123. The button caption changes to Restore Disconnected Recordset and the Send Batch Updates to Server button is enabled. The new values don't appear in the datasheet because the updateBatch method hasn't been applied at this point.
4. Click the Send Batch Updates to Server button to reopen the connection, execute the updateBatch method, and close the connection. The datasheet displays the updated Recordset returned by the server (see Figure 30.20).

**Figure 30.20**

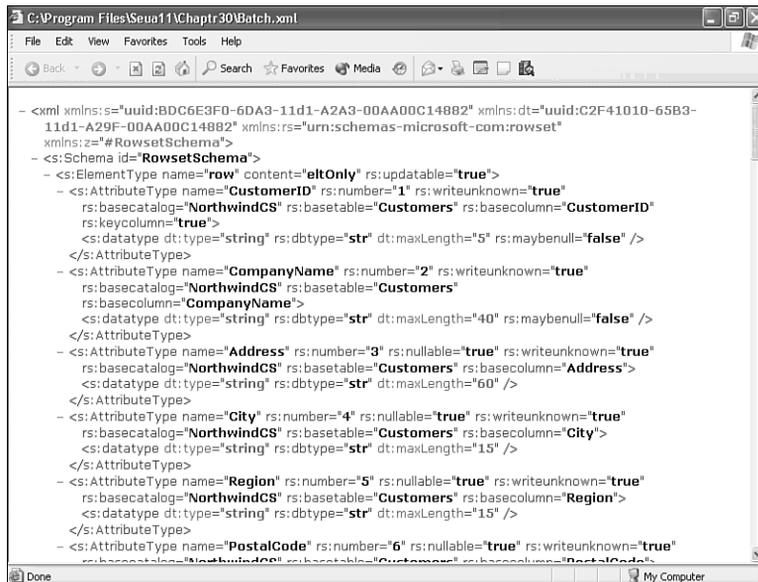
Clicking Send Batch Updates to Server updates the server table and then retrieves the updated Recordset by opening and then closing another connection.



5. Click Restore Disconnected Recordset and Send Batch Updates to Server to return the Customers table to its original state. (Clicking Update and Restore Disconnected Recordset toggles the Region values in the local Recordset.)
6. Click the Open Batch.xml in IE 5+ button to launch IE with `file://path/Batch.xml` as the URL. IE 5+'s XML parser formats the attribute-centric document and color-codes XML tags (see Figure 30.21).

**Figure 30.21**

IE 5+'s XML parser formats the XML document saved as Batch.xml. This example shows the Schema elements collapsed and the data elements expanded. (The Region attribute of the row element is in the updated state.)



If you update the local copy of the Recordset and don't send the changes to the server, you receive a message reminding you that changes are pending when you close the form. If you don't save the changes to the server and reopen the form, a message asks if you want to send the changes to the server before proceeding.

## VBA CODE IN THE frmBatchUpdate CLASS MODULE

The VBA code of the event-handling and supporting subprocedures of the frmBatchUpdate Class Module illustrates how to program many of the ADO properties and methods described in the preceding sections devoted to the Connection and Recordset objects. The Command object isn't used in this example, because the form opens Recordset objects on a temporary Connection object or from a copy of a Recordset persisted to a local file in ADTG format.

30

### THE Form\_Load EVENT HANDLER

Listing 30.3 shows the VBA code for the Form\_Load event handler. The first operation uses the VBA `Dir` function to determine whether the Batch.rst file exists; if so, response to the message specified by the `MsgBox` function determines whether existing updates are processed by the `cmdUpdate_Click` subprocedure or discarded.

#### LISTING 30.3 CODE FOR SAVING THE INITIAL Recordset OBJECT

```
Private Sub Form_Load()
    'Open the connection, and create and display the Recordset

    blnUseJet = False 'Set True to use the Jet provider

    'Test for presence of the saved Recordset
    If Dir(CurrentProject.Path & "\Batch.rst") <> "" Then
        'File is present so updates are pending
        If MsgBox("Do you want to send your changes to the server?", vbQuestion +
vbYesNo, _
            "Updates Are Pending for the Server") = vbYes Then
            Call cmdUpdate_Click
            Exit Sub
        Else
            Kill CurrentProject.Path & "\Batch.rst"
        End If
    End If

    'Create a Form object variable for the subform
    Set sbfBatch = Forms!frmBatchUpdate!sbfBatchUpdate.Form
    Me.cmdBulkUpdate.SetFocus
    Me.cmdUpdate.Enabled = False
    Me.cmdOpenXML.Enabled = False

    'Open a connection to the server
    Call OpenConnection

    'Create a Recordset for Batch Updates
    strSQL = "SELECT CustomerID, CompanyName, Address, City, Region, PostalCode,
Country FROM Customers"
    With rstBatch
        Set .ActiveConnection = cnnBatch
        .CursorType = adOpenStatic
        .CursorLocation = adUseClient
    End With
End Sub
```

```

.LockType = adLockBatchOptimistic
.Open strSQL

'Save the Recordset to a file
.Save CurrentProject.Path & "\Batch.rst", adPersistADTG

'Save an XML version
On Error Resume Next
Kill CurrentProject.Path & "\Batch.xml"
.Save CurrentProject.Path & "\Batch.xml", adPersistXML
On Error GoTo 0
Me.cmdOpenXML.Enabled = True

'Disconnect the Recordset
Set .ActiveConnection = Nothing

If .Fields("Region").Value = "123" Then
    Me.cmdBulkUpdate.Caption = "Restore Disconnected Recordset"
Else
    Me.cmdBulkUpdate.Caption = "Update Disconnected Recordset"
End If
End With

'Destroy the connection
cnnBatch.Close
Set cnnBatch = Nothing

'Open a local Recordset from the saved file
Call OpenRstFromFile

'Delete the source of the file Recordset
Kill CurrentProject.Path & "\Batch.rst"
Me.Caption = "Datasheet Contains Values from Server (Disconnected Recordset)"
End Sub

```

**NOTE**

In a real-world application, you probably wouldn't delete a saved Recordset that contains updates. Instead of deleting the file with a **Kill** instruction, you would open the saved Recordset to permit continued editing.

The **Set** `sbfBatch = Forms!frmBatchUpdate!sbfBatchUpdate.Form` statement creates a Form object for the subform, so you can set property values for the `sbfBatchUpdate` subform by code of the `frmBatchUpdate` form in the `OpenRstFromFile` subprocedure. Combining the VBA code for forms and subforms in a single Class Module makes the code more readable.

→ For more information on the strange syntax to point to another Form or Report object, see "Referring to Access Objects with VBA," p. 1218.

After disabling the Send Updates to Server and Open Batch.xml in IE 5+ buttons, the code calls the `OpenConnection` subprocedure to create a temporary Connection object, creates a

Recordset object with batch-optimistic locking, saves the Recordset to Batch.rst and Batch.xml, and disconnects the Recordset from the connection with the **Set .ActiveConnection = Nothing** statement. Finally the code closes the Connection, releases it from memory, calls the `OpenRstFromFile` subprocedure, and deletes the Batch.rst file.

### THE `OpenConnection` SUBPROCEDURE

The `OpenConnection` subprocedure (see Listing 30.4) accommodates a Jet database by setting the value of `blnUseJet` to `True` in the `Form_Load` event handler. By default, the code attempts to open the connection with integrated Windows security. If this attempt fails, the code attempts to use SQL Server security with the `sa` logon ID (`UID=sa`) and no password. (If you've secured the `sa` account, add the password for the account to `PWD=.`)

#### LISTING 30.4 CONNECTING TO A JET DATABASE OR USE SQL SERVER OR INTEGRATED WINDOWS SECURITY TO CONNECT TO THE LOCAL MSDE INSTANCE

```
Private Sub OpenConnection()
    'Specify the OLE DB provider and open the connection
    With cnnBatch
        If blnUseJet Then
            .Provider = "Microsoft.Jet.OLEDB.4.0"
            .Open CurrentProject.Path & "\Northwind.mdb", "Admin"
        Else
            On Error Resume Next
            'Try integrated Windows security first
            .Open "Provider=SQLOLEDB.1;Data Source=(local);" & _
                "Integrated Security=SSPI;Initial Catalog=NorthwindCS"
            If Err.Number Then
                Err.Clear
                On Error GoTo 0
                'Now try SQL Server security
                .Open "Provider=SQLOLEDB.1;Data Source=(local);" & _
                    "UID=sa;PWD=;Initial Catalog=NorthwindCS"
            End If
        End If
    End With
End Sub
```

### THE `OpenRstFromFile` SUBPROCEDURE

The code for the `OpenRstFromFile` Subprocedure derives from that behind the `frmADO_Jet` and `frmADO_MSDE` forms. The primary difference in the code of Listing 30.5 is that the `Recordset.Open` method specifies the temporary `Batch.rst` file as its data source.

#### LISTING 30.5 OPENING A SAVED Recordset OBJECT AND ASSIGNING IT TO THE Recordset PROPERTY OF THE SUBFORM

```
Private Sub OpenRstFromFile()
    If rstBatch.State = adStateOpen Then
        rstBatch.Close
    End If
```

```

rstBatch.Open CurrentProject.Path & "\Batch.rst", , adOpenStatic, _
    adLockBatchOptimistic, adCmdFile
With sbfBatch
    'Assign rstBatch as the Recordset for the subform
    Set .Recordset = rstBatch
    .UniqueTable = "Customers"
    .txtCustomerID.ControlSource = "CustomerID"
    .txtCompanyName.ControlSource = "CompanyName"
    .txtAddress.ControlSource = "Address"
    .txtCity.ControlSource = "City"
    .txtRegion.ControlSource = "Region"
    .txtPostalCode.ControlSource = "PostalCode"
    .txtCountry.ControlSource = "Country"
End With
End Sub

```

### THE cmdBulkUpdate EVENT HANDLER

Clicking the Update Disconnected Recordset/Restore Disconnected Recordset button executes the cmdBulkUpdate event-handler (see Listing 30.6). The **Set** sbfBatch.Recordset = **Nothing** statement prevents flashing of the subform during edits performed in the **Do While Not .EOF...Loop** process. This loop traverses the Recordset and changes the values of unused Region cells from NULL to 123 or vice-versa. After the loop completes, the form hooks back up to the edited Recordset. The call to the Form\_Load subprocedure displays the updated Customers table fields in the subform.

#### NOTE

Real-world applications use an unbound form and unbound text boxes to edit the Recordset. The form requires command buttons to navigate the Recordset by invoking Move... methods. The event handler for an Update Record button makes the changes to the field values of the local Recordset.

### LISTING 30.6 THE cmdBulkUpdate EVENT HANDLER USES A LOOP TO EMULATE MULTIPLE RECORDSET EDITING OPERATIONS

```

Private Sub cmdBulkUpdate_Click()
    Dim blnUpdate As Boolean
    Dim strCapSuffix As String

    'Housekeeping for form and button captions
    strCapSuffix = " While Disconnected (Updates Are Pending)"
    If Me.cmdBulkUpdate.Caption = "Update Disconnected Recordset" Then
        Me.Caption = "Changing Empty Region Values to 123" & strCapSuffix
        blnUpdate = True
    Me.cmdBulkUpdate.Caption = "Restore Disconnected Recordset"
    Else
        Me.Caption = "Returning Region Values from 123 to Null" & strCapSuffix
        blnUpdate = False
    Me.cmdBulkUpdate.Caption = "Update Disconnected Recordset"
    End If

```

*continues*

```
'If you don't execute the following instruction, the subform
'datasheet can cause flutter vertigo during updates
Set sbfBatch.Recordset = Nothing

'Set the Field variable (improves performance)
Set fldRegion = rstBatch.Fields("Region")

'Now update or restore Region values
With rstBatch
  .MoveFirst
  Do While Not .EOF
    If blnUpdate Then
      If IsNull(fldRegion.Value) Then
        fldRegion.Value = "123"
      End If
    Else If
      'Restore the original Null value
      If fldRegion.Value = "123" Then
        fldRegion.Value = Null
      End If
    End If
  .MoveNext
Loop
On Error Resume Next
'For safety
Kill CurrentProject.Path & "\Batch.rst"
On Error GoTo 0
  .Save CurrentProject.Path & "\Batch.rst", adPersistADTG
End With

'Now restore the subform's Recordset property
Set sbfBatch.Recordset = rstBatch
Me.cmdUpdate.Enabled = True
End Sub
```

**TIP**

Create a Field variable (fldRegion), instead of using a Recordset.Fields(strFieldName).Value = varValue instruction. Specifying a Field variable improves performance, especially if the Recordset has many fields.

**THE cmdUpdate EVENT HANDLER**

Clicking the Send Updates to Server button executes the cmdUpdate event handler and the UpdateBatch method to update the server tables (see Listing 30.7). Before executing the update, **Debug.Print** statements record the OriginalValue and Value property values for the first row in the Immediate window.

### LISTING 30.7 UPDATING THE SERVER TABLES RECONNECTS THE Recordset TO THE DATA SOURCE, EXECUTES THE UpdateBatch METHOD, AND CLOSES THE Connection

```

Private Sub cmdUpdate_Click()
    'Recreate the connection
    Call OpenConnection

    'Reopen the Recordset from the file
    With rstBatch
        If .State = adStateOpen Then
            .Close
        End If
        Set rstBatch.ActiveConnection = cnnBatch
        .Open CurrentProject.Path & "\Batch.rst", , adOpenStatic, _
            adLockBatchOptimistic, adCmdFile

        'To demonstrate these two properties
        Debug.Print "Original Value: " & .Fields("Region").OriginalValue
        Debug.Print "Updated Value: " & .Fields("Region").Value

        'Send the updates to the server
        .UpdateBatch
        .Close
    End With

    'Clean up
    Set rstBatch = Nothing
    cnnBatch.Close
    Set cnnBatch = Nothing
    Kill CurrentProject.Path & "\Batch.rst"

    'Load the subform datasheet from the server
    Call Form_Load
    Me.Caption = "Updated Values Retrieved from Server"
End Sub

```

### THE cmdOpenXML EVENT HANDLER

The cmdOpenXML event handler for the Open Batch.rst in IE 5+ button demonstrates use of the VBA **Shell** function to launch another application (see Listing 30.8). The argument of the **Shell** function is identical to the instruction you type in the Run dialog's Open text box to launch an application manually. If successful, the **Shell** function returns the task ID value of the running application; if not, the function returns an empty **Variant** value.

### LISTING 30.8 OPENING A PERSISTENT Recordset OBJECT SAVED AS AN XML FILE IN IE 5+

```

Private Sub cmdOpenXML_Click()
    'Launch IE 5+ with Batch.xml as the source URL
    Dim strURL As String
    Dim strShell As String
    Dim varShell As Variant

```

*continues*

**LISTING 30.8 CONTINUED**

```

strURL = "file://" & CurrentProject.Path & "\Batch.xml"
strShell = "\Program Files\Internet Explorer\Iexplore.exe " & strURL
varShell = Shell(strShell, vbNormalFocus)
If IsEmpty(varShell) Then
    MsgBox "Can't open Internet Explorer", vbOKOnly + vbExclamation, _
        "Unable to Display Batch.xml"
End If
End Sub

```

**THE Form\_Unload EVENT HANDLER**

Variables in form Class Modules disappear (go out of scope) when the form closes. However, it's a good programming practice to "clean up" all object variables before closing a form. In addition to cleanup operations, this event handler (see Listing 30.9) detects the presence of unsent updates in Batch.rst. Setting the `intCancel` argument to **True** cancels the unload operation.

**LISTING 30.9 THE Form\_Unload EVENT HANDLER CHECKS FOR UNSENT UPDATES AND, IF THE USER CLICKS YES IN THE MESSAGE BOX, CLOSSES OPEN OBJECTS AND SETS THEM TO Nothing**

```

Private Sub Form_Unload(intCancel As Integer)
    'Check for pending updates before unloading
    If Dir(CurrentProject.Path & "\Batch.rst") <> "" Then
        If MsgBox("Are you sure you want to quit now?", vbQuestion + vbYesNo, _
            "Updates Are Pending for the Server") = vbNo Then
            intCancel = True
        Exit Sub
    End If
End If
    'Clean up objects
    If rstBatch.State = adStateOpen Then
        rstBatch.Close
    End If
    Set rstBatch = Nothing
    If cnnBatch.State = adStateOpen Then
        cnnBatch.Close
    End If
    Set cnnBatch = Nothing
    'If you don't execute the following instruction,
    'you receive an error when opening the form
    Set sbfBatch.Recordset = Nothing
End Sub

```

## TIP

Unlike Visual Basic forms, values you assign with VBA code to Access Form, Report, and Control objects persist after closing the object and exiting the Access application. In some cases, reopening the object results in an error message. Executing the `Set sbfBatch.Recordset = Nothing` instruction before closing the form and its subform prevents the possibility of an error on reopening the form, because the source value of the `Recordset` property isn't present before the `Form_Load` event handler executes.

## PROGRAMMING Stream OBJECTS



For Access programmers, Stream objects primarily are of interest for returning attribute-centric XML data documents from SQL Server 2000. The T-SQL statement for the query must terminate with the `FOR XML AUTO` or `FOR XML RAW` option. Both options return a well-formed XML document using Microsoft's `xml-sql` schema. Unlike the .xml files saved from Recordset objects with the `adPersistXML` option, the stream doesn't include the schema elements. Like the rowset schema, `xml-sql` isn't compatible with Access 2003's native XML schema. SQL Server HTTP template queries, which can return HTML tables to Web browsers from `FOR XML AUTO` queries, require the `xml-sql` schema.

→ For an example of using the `FOR XML AUTO` option in SQL Server HTTP template queries, see "Using SQL Server 2000's HTTP Query Features," p. 976.

### EXECUTING FOR XML AUTO QUERIES WITH THE frmSTREAM FORM

The `frmStream` form has unbound text boxes to display a default T-SQL `FOR XML AUTO` query, the modifications to the query syntax needed to return a well-formed XML document, and the XML document resulting from execution of the `Command` object that specifies `MSSQLXML` as the query dialect. To test the `frmStream` form, do this:



1. Open `ADOTest.mdb`'s or `ADOTest.adp`'s `frmStream` form. The default query is a simple T-SQL query, similar to that used by the `frmBatchUpdate` form, with the `FOR XML AUTO` modifier added. SQL Server's default rowset document style is attribute-centric. Mark the `Element-Centric` check box to add `ELEMENTS` to the modifier and return an element-centric document.
2. Click the `Execute FOR XML Query` button to display the XML query wrapper required by SQL Server 2000 to return a well-formed XML data document. A `Command` object returns a `Stream` object that contains an XML data document, which opens in the bottom text box. The `Stream` object is saved to `Stream.xml` in the folder that contains `ADOTest.mdb`.
3. Click the `Open Stream.xml in IE 5+` button to launch IE with `file://path/Stream.xml` as the URL. IE's XML parser makes it easier to read the XML document.

4. Mark the Multi-Table Query check box to replace the simple query with a T-SQL query against the Customers and Orders tables. Making a change to the T-SQL FOR XML Query text box clears the other two text boxes.
5. Click Execute FOR XML Query again to display the resulting XML document (see Figure 30.22).

**Figure 30.22**

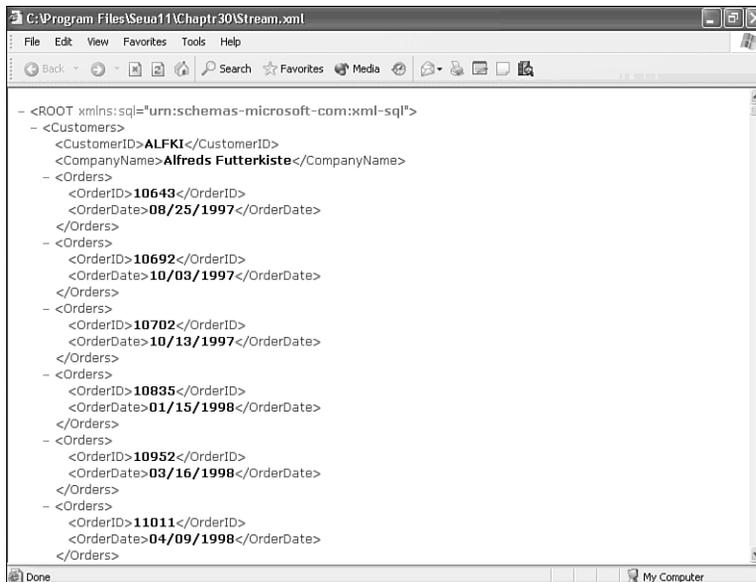
The multi-table query with the FOR XML AUTO, ELEMENTS option returns elements from the Orders table nested within Customers table elements.



6. Click Open Stream.xml in IE 5+. The nesting of Orders elements within the Customers is more evident in IE's presentation (see Figure 30.23).

**Figure 30.23**

IE 5+'s XML parser formats the document to make nesting of table elements readily apparent.



7. To see the effect of the FOR XML RAW modifier, replace AUTO with RAW in the T-SQL query, execute the command, and open the query in IE 5+ (see Figure 30.24).

**Figure 30.24**

The FOR XML RAW modifier combines all attribute values for a query row in a single, generic row element.

```

<?xml:namespace prefix="sql" uri="urn:schemas-microsoft-com:xml-sql">
<row CustomerID="ALFKI" CompanyName="Alfreds Futterkiste" OrderID="10643"
OrderDate="08/25/1997" />
<row CustomerID="ALFKI" CompanyName="Alfreds Futterkiste" OrderID="10692"
OrderDate="10/03/1997" />
<row CustomerID="ALFKI" CompanyName="Alfreds Futterkiste" OrderID="10702"
OrderDate="10/13/1997" />
<row CustomerID="ALFKI" CompanyName="Alfreds Futterkiste" OrderID="10835"
OrderDate="01/15/1998" />
<row CustomerID="ALFKI" CompanyName="Alfreds Futterkiste" OrderID="10952"
OrderDate="03/16/1998" />
<row CustomerID="ALFKI" CompanyName="Alfreds Futterkiste" OrderID="11011"
OrderDate="04/09/1998" />
<row CustomerID="ANATR" CompanyName="Ana Trujillo Emparedados y helados" OrderID="10926"
OrderDate="03/04/1998" />
<row CustomerID="ANATR" CompanyName="Ana Trujillo Emparedados y helados" OrderID="10759"
OrderDate="11/28/1997" />
<row CustomerID="ANATR" CompanyName="Ana Trujillo Emparedados y helados" OrderID="10625"
OrderDate="08/08/1997" />
<row CustomerID="ANATR" CompanyName="Ana Trujillo Emparedados y helados" OrderID="10308"
OrderDate="09/18/1996" />
<row CustomerID="ANTON" CompanyName="Antonio Moreno Taqueria" OrderID="10365"
OrderDate="11/27/1996" />
<row CustomerID="ANTON" CompanyName="Antonio Moreno Taqueria" OrderID="10573"
OrderDate="06/19/1997" />
<row CustomerID="ANTON" CompanyName="Antonio Moreno Taqueria" OrderID="10507"
OrderDate="04/15/1997" />
<row CustomerID="ANTON" CompanyName="Antonio Moreno Taqueria" OrderID="10535"
OrderDate="05/13/1997" />

```

## NOTE

Changing the ORDER BY clause from Customers.CustomerID to Orders.OrderID generates a very different XML document structure. In this case, most Customers elements contain a single nested order; only consecutive orders for a particular customer appear as multiple nested order elements. (See the entry for ROMEY as the first example.)

## EXPLORING THE VBA CODE TO CREATE A Stream OBJECT

Most of the event handlers and subprocedures used by the VBA code for the frmStream form derive from those of the frmBatch form described earlier. The two important code elements behind frmStream are the Declarations section, which declares the ADODB.Command and ADODB.Stream object variables, and constants for the currently allowable GUID values of the Command.Dialect property, and the cmdExecute\_Click event handler (see Listing 30.10).

### LISTING 30.10 CREATING A STREAM OBJECT FROM AN SQL SERVER FOR XML AUTO QUERY AND DISPLAYING THE STREAM IN A TEXT BOX

**Option** Compare Database  
**Option** Explicit

*continues*

## LISTING 30.10 CONTINUED

```

Private cnnStream As New ADODB.Connection
Private cmmStream As New ADODB.Command
Private stmQuery As ADODB.Stream

'GUID constants for Stream.Dialect
Private Const DBGUID_DEFAULT As String = _
    "{C8B521FB-5CF3-11CE-ADE5-00AA0044773D}"
Private Const DBGUID_SQL As String = _
    "{C8B522D7-5CF3-11CE-ADE5-00AA0044773D}"
Private Const DBGUID_MSSQLXML As String = _
    "{5D531CB2-E6Ed-11D2-B252-00C04F681B71}"
Private Const DBGUID_XPATH As String = _
    "{ec2a4293-e898-11d2-b1b7-00c04f680c56}"

'Constants for XML query prefix and suffix
Private Const strXML_SQLPrefix As String = _
    "<ROOT xmlns:sql='urn:schemas-microsoft-com:xml-sql'>" & vbCrLf & "<sql:query>"
Private Const strXML_SQLSuffix As String = "</sql:query>" & vbCrLf & "</ROOT>"

Private Sub cmdExecute_Click()
    'Use Command and Stream objects to return XML as text
    Dim strXMLQuery As String
    Dim strXML As String
    Dim lngCtr As Long

    On Error GoTo errGetXMLStream
    strXMLQuery = Me.txtQuery.Value

    'Add the XML namespace and <ROOT...> and </ROOT> tags to the query text
    strXMLQuery = strXML_SQLPrefix & vbCrLf & strXMLQuery & vbCrLf &
    strXML_SQLSuffix

    'Display the CommandText property value
    Me.txtXMLQuery.Value = strXMLQuery
DoEvents

    'Create a new Stream for each execution
    Set stmQuery = New ADODB.Stream
    stmQuery.Open

    'Set and execute the command to return a stream
    With cmmStream
        Set .ActiveConnection = cnnStream
        'Query text is used here, not an input stream
        .CommandText = strXMLQuery
        'Specify an SQL Server FOR XML query
        .Dialect = DBGUID_MSSQLXML
        'Specify the stream to receive the output
        .Properties("Output Stream") = stmQuery
        .Execute , , adExecuteStream
    End With

    'Reset the stream position
    stmQuery.Position = 0

```

```

'Save the stream to a local file
stmQuery.SaveToFile CurrentProject.Path & "\Stream.xml", adSaveCreateOverWrite
cmdOpenXML.Enabled = True

'Extract the text from the stream
strXML = stmQuery.ReadText

'Make the XML more readable with line feeds, if it isn't too long
If Len(strXML) < 15000 Then
    Me.txtXML.Value = Replace(strXML, "><", ">" & vbCrLf & "<")
Else
    If Len(strXML) > 32000 Then
        'Limit the display to capacity of text box
        Me.txtXML.Value = Left$(strXML, 30000)
    Else
        Me.txtXML.Value = strXML
    End If
End If
Exit Sub

errGetXMLStream:
    MsgBox Err.Description, vbOKOnly + vbExclamation, "Error Returning XML Stream"
    Exit Sub
End Sub

```

This form only uses the `DBGUID_MSSQLXML` constant; the other three GUID constants are for reference only. ADO 2.6+'s type library doesn't have a "DialectGUIDEnum" or similar enumeration, so you must declare at least the `DBGUID_MSSQLXML` constant to request SQL Server to return XML data documents in the `xml-sql` dialect. Comments in the body of the code of the `cmdExecute_Click` event handler describe the purpose of each Stream-related statement.

## EXPLORING THE ADDORDER.ADP SAMPLE PROJECT



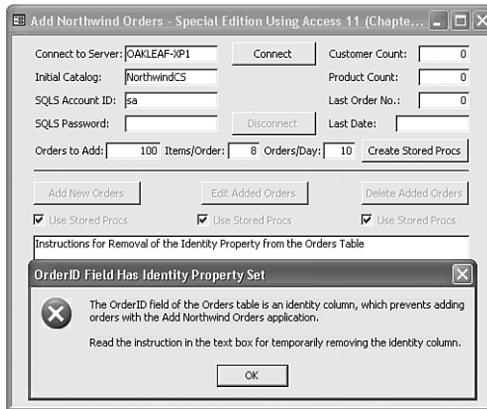
The `AddOrders.adp` sample project in the `\Seua11\Chaptr30` folder of the accompanying CD-ROM demonstrates practical application for this chapter's example of the programming of ADO objects, methods, and properties. The primary purpose of the `AddOrders` project is to add a large number of records to the `Orders` and `Order Details` tables of a copy of the `NorthwindCS` SQL Server database. Working with test tables having a large number of rows lets you debug online transaction processing (OLTP) applications with real-world data. The code uses random record numbers to specify the `CustomerID` for each added order and the `ProductID` for order line items.

The `AddOrders` project also lets you compare the performance difference between sending SQL statements to the server and using stored procedures to add, edit, and delete `Orders` and `Order Details` records. Code in the `frmAddNorthwindOrders` Class Module creates the required stored procedures.

To give the AddOrders.adp project a test drive, do the following:

1. Open NorthwindCS.adp, choose **T**ools, **D**atabase Tools, **T**ransfer Database and create a copy of NorthwindCS as NorthwindSQL or any other name you prefer in your (local) SQL Server instance.
2. Open AddOrders.adp, choose **F**ile, **C**onnection, and specify the server, authentication method, and name of the database copy. When you add a connection to AddOrders.adp, you have the option of using the project's connection or specifying a connection to another server or database.
3. Click **C**onnect and click **Y**es when the message box asks if you want to use the current connection to the new database. If you specified SQL Server security, type your login ID and password in the two text boxes. You receive the message about the OrderID field's *Identity* attribute shown in Figure 30.25.

**Figure 30.25**  
Opening NorthwindCS or a copy of NorthwindCS displays the warning message shown here.

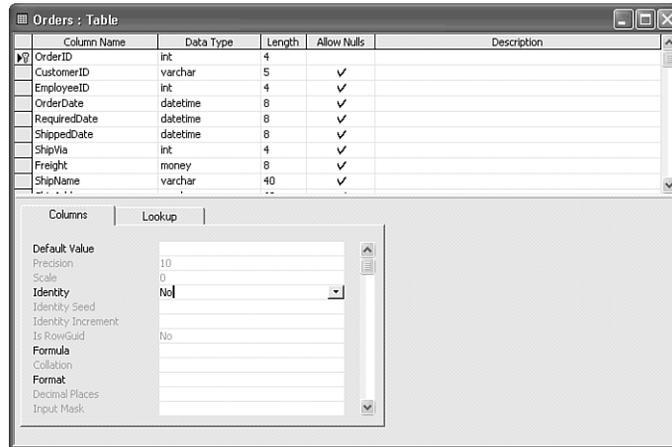


#### NOTE

The *Identity* attribute must be removed from the Orders table to permit deleting added records and then adding new records with numbers that are consecutive with the original OrderID numbers (10248–11077).

4. Acknowledge the message, open the Orders table in Design view, and select the OrderID column. In the Columns property page, select *Identity* and change the value from **Y**es to **N**o (see Figure 30.26). Close the table and save the changes.

**Figure 30.26**  
Set the Identity attribute of the OrderID field of the Orders table to No in the daVinci table designer.

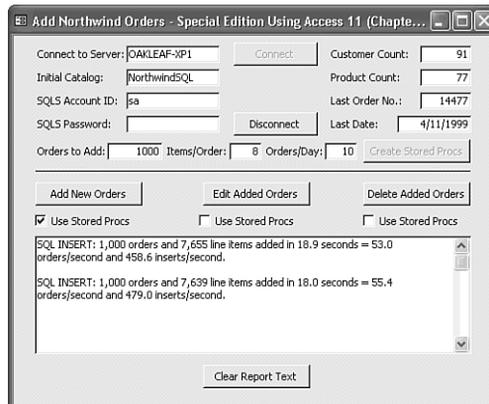


### TIP

You also can change the design in SQL Server Enterprise Manager, but using Access's Table Design view is easier.

- In the form, click the Clear Report Text button, click Connect to open the connection to the database, and click the Create Stored Procs button to add three stored procedures to the database. Adding the procedures enables the three Use Stored Procs check boxes.
- Change the Orders to Add, Items/Order, and Orders/Day values, if you want, and click Add New Orders. The number of Orders and Order Details records added and the time required for addition appears in the text box.
- Mark the Use Stored Procs check box under the Add New Orders button to compare the speed of order addition with a stored procedure. Depending on your hardware configuration, you might gain about a 10% performance improvement by using a stored procedure instead of sending INSERT statements to the database (see Figure 30.27).

**Figure 30.27**  
Adding, editing, or deleting records adds the number of records affected and timing data to the text box.



- Repeat steps 6 and 7, but click the Edit Added Orders button to change the Quantity values of all Order Details records you've added.
- Click the Delete Added Orders button to restore the tables to their original number of records. A message box lets you choose between bulk and individual order deletion.

**NOTE**

The code in the form's Class Module originated in a Visual Basic 6.0 program for testing SQL Server 7.0 and 2000 OLTP performance in a variety of server hardware configurations. Only a few code changes were necessary to the Visual Basic 6.0 code that was copied to the Access form.

## TROUBLESHOOTING



### SPACES IN ADO OBJECT NAMES

*When I attempt to open a Command object on the views, functions, or stored procedures of NorthwindCS, I receive a “Syntax error or access violation” message.*

SQL Server 7.0+ (unfortunately) supports spaces in object names, such as views and stored procedures. However, SQL Server wants these names enclosed within double quotes. Sending double quotes in an object name string is a pain in VBA, but surrounding the object name with square brackets also solves the problem. For example, `cnnName.CommandText = "Sales By Year"` fails but `cnnName.CommandText = "[Sales By Year]"` works. Using square brackets for otherwise-illegal object identifiers is the better programming practice.

## IN THE REAL WORLD—WHY LEARN ADO PROGRAMMING?

As observed in Chapter 4, “Exploring Relational Database Theory and Practice,” “Everything has to be somewhere” is a popular corollary of the Law of Conservation of Matter. So just about everything you need to know about ADO 2.x and OLE DB is concentrated in this chapter. The problem with this “laundry list” approach to describing a set of data-related objects is that readers are likely to doze off in mid-chapter. If you’ve gotten this far (and have at least scanned the intervening code and tables), you probably surmised that ADO is more than just a replacement for DAO—it’s a relatively new and expansive approach to database connectivity.

The most important reason to become an accomplished ADO programmer is to create Web-based database applications. Microsoft designed OLE DB and ADO expressly for HTML- and XML-based applications, such as DAP—the subject of the three chapters of Part VI, “Publishing Data to Intranets and the Internet.” You can use VBScript or JScript (Microsoft’s variant of ECMAScript) to open and manipulate ADO `Connection`, `Command`,

and `Recordset` objects on Web pages. With DAO, you're stuck with conventional Access applications that require users to have a copy of Office 2003 or you to supply runtime versions of your Access 2003 applications.

Another incentive for becoming ADO-proficient is migrating from Jet 4.0 to ADP and SQL Server back ends. When SQL Server marketing honchos say that SQL Server is Microsoft's "strategic database direction," believe them. Jet still isn't dead, but the handwriting is on the wall; ultimately SQL Server will replace Jet in all but the most trivial database applications. The ADO 2.7 documentation on MSDN states that "Microsoft has deprecated the Microsoft Jet Engine, and plans no new releases or service packs for this component." SQL Server 2000 Standard Edition and MSDE 2000 dominate the "sweet spot" of the client/server RDBMS market—small- to mid-size firms—plus division-level intranets of the Fortune 1000. SQL Server Enterprise and DataCenter editions are making inroads on Oracle's and IBM's share of the enterprise RDBMS market.

#### NOTE

Microsoft's intention might have been to release no new service packs (SPs) for Jet 4.0, but Access 2003 required a new Jet 4.0 SP7 to support macro security and "sandbox" mode.

Microsoft released .NET Framework 1.0 and Visual Studio .NET on February 13, 2002. ADO.NET now is Microsoft's strategic data access approach. Word and Excel are the only members of Office 2003 to integrate Visual Basic .NET, C#, J#, and other .NET Common Language Runtime-compliant programming languages as alternatives to VBA. As mentioned in earlier chapters, moving from VBA to object-oriented programming with Visual Basic .NET is challenging for most Office developers and overwhelming for Access power users. It's a good bet that VBA will dominate Office-related programming for at least the next five years.

.NET Framework 1.0 and 1.1 include managed (native) .NET data providers only for SQL Server and Oracle 7.3x databases, and managed wrappers for OLE DB and ODBC. (The Oracle provider offers "limited support for Oracle 8x.") IBM offers a .NET managed provider for DB2. Until other RDBMS vendors—such as MySQL AB and Sybase—write native .NET providers for their databases, Visual Studio .NET programmers must rely on OLE DB, ADO and .NET's COM interoperability (COM interop) layer to connect to other popular RDBMSs. Thus, OLE DB and ADO are likely to remain in widespread use through at least the first decade of the twentieth century.

The ultimate answer to "Why learn ADO programming?" however, is "Microsoft wants YOU to use ADO until you move to .NET." Like the proverbial one-ton gorilla, Microsoft usually gets what it wants, but it will take more than chest-thumping and bellowing to convince today's VBA programmers to take on the challenge of Visual Basic .NET and ADO.NET.