CHAPTER 11

The ADO Object Model

In this part of the book, I'm going to turn my attention from the tools that can make it easier to build your applications and focus on the underlying object model that they use. Over time, I found that using the ADO objects directly is often easier than using the tools. In this chapter, I'm going to present a brief overview of the ADO object model and explain how the various objects fit together. In the rest of Part III, I'll dig into each of the key objects and show you how easy they are to use.

In This Chapter

Introducing ADO 2.5

Using ADO objects

Working with the ADO Errors collection

Introducing ActiveX Data Objects 2.5

The ActiveX Data Objects (ADO) is Microsoft's way of implementing *Universal Data Access*. Universal Data Access allows you to use the same high-speed interface for both relational and non-relational data, while providing an easy-to-use, language-independent interface.

The ADO object model

ADO carries over some of the objects used in the older DAO and RDO object models discussed in Chapter 6. However, it uses a completely different approach that removes the strict hierarchy required when using the older object models (see Figure 11-1 and Table 11-1).

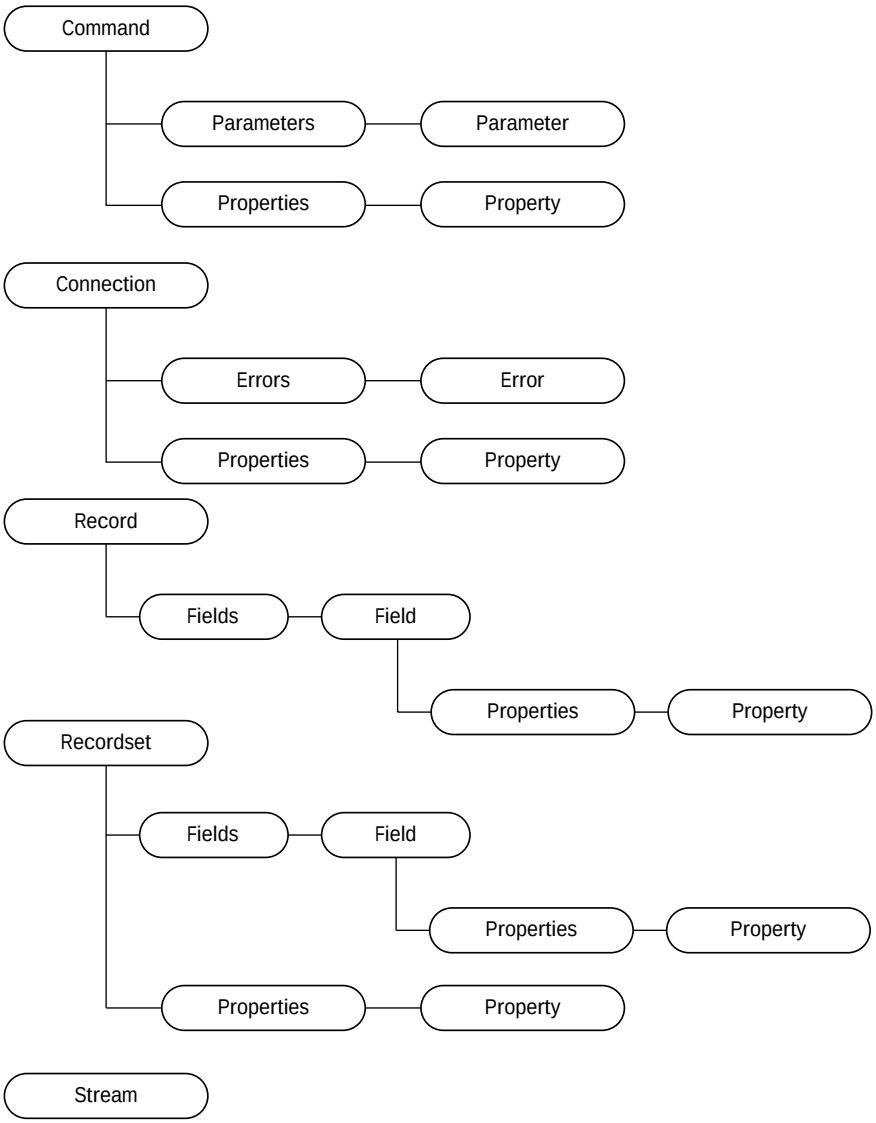

Figure 11-1: Presenting the ADO object model

There are three main objects in the ADO object model that are used for database access. The `Connection` object provides a path to the data source. The `Command` object contains the information necessary to execute an SQL Statement or a stored procedure. The `Recordset` object contains the results from a query. The `Command` and `Recordset` objects have the ability to create an implicit `Connection` object if desired, so you can access a set of records or perform a database function by creating only a single object.

Table 11-1
ADO Objects

Object	*Description*
`Command`	Executes an SQL Statement or stored procedure.
`Connection`	Used to manage the information necessary to connect to a database or other OLE DB data provider.
`Error`	Contains information about a specific error.
`Errors`	Contains a collection of `Error` objects.
`Field`	Contains information about a specific field in the database.
`Fields`	Contains a collection of `Field` objects.
`Parameter`	Holds information that is passed to or returned from a stored procedure or parameterized query.
`Parameters`	Contains a collection of `Parameter` objects.
`Property`	Contains a dynamic property that is defined by the OLE DB data provider.
`Properties`	Contains a collection of `Property` objects.
`Record`	Represents a row in `Recordset` object or a file or an e-mail message.
`Recordset`	Used to manage the set of rows generated by a query operation.
`Stream`	Represents a stream of binary data or text.

The `Record` and `Stream` objects are used primarily to support access to non-database resources. The `Record` object represents either a row in a `Recordset` or a file or an e-mail message. A `Stream` object provides the facilities to manipulate the data in a file or an e-mail message.

New in ADO 2.5

If you are familiar with the previous versions of ADO, here's a quick introduction to the new features.

The Record object

The `Record` object is used to represent information such as directories and files in a file system, and folders and messages in an e-mail system. A `Record` object can also be used to represent a row from a `Recordset` object.

The Stream object

The Stream object is used to read and write binary information to and from files and messages associated with the Record object.

URLs

URLs may now be used in place of connection strings (see the Connection object) and command text (see the Command object) as an alternative to using traditional connection strings and command text.

New properties and methods

Several new properties and methods have been added to the ADO library. The Mode property is available in the Connection, Record, and Stream objects and describes the permissions available for modifying data. The rest of the new properties and methods are used by the new Stream and Recordset objects to manage files and e-mail messages.

Introducing ActiveX Data Objects Extensions

While not exactly new in ADO 2.5, the ActiveX Data Objects Extensions (ADOX) gives you the capability to create and modify a database's structure (also known as a *schema*) and maintain security through an object-oriented approach. Since these objects are independent of any single data provider, it is possible to build a general-purpose program that would work with any data provider.

The ADOX object model (see Figure 11-2 and Table 11-2) was developed as an object-oriented way for someone to access a database schema. The schema provides access to all of the possible objects in a database: tables, indexes, stored procedures and views, as well as security information.

The Catalog is the root object. All other objects can be referenced from the Catalog. Beneath the Catalog object are the five main collections of database objects, the Tables collection, the Groups collection, the Users collection, the Procedures collection, and the Views collection.

The Tables collection documents the physical structure of the data. From the Table object, you can find out the details of the structure by examining the Columns, Indexes, and Keys collections. Note that the Index and Key objects also reference the Columns collection. However, before you can add a Column object to either the Index or Key object, it must already exist in the Table object.

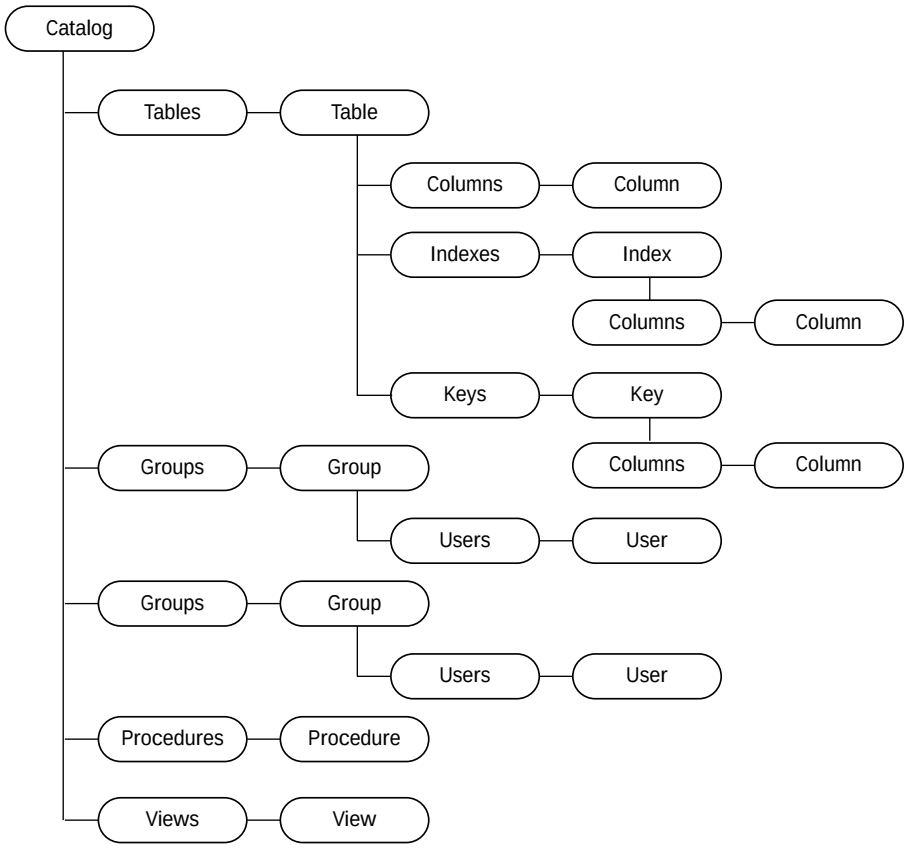

Figure 11-2: Presenting the ADOX object model

Table 11-2
ADOX Objects

Object	*Description*
`Catalog`	Contains the schema definition of the data source. This is the base object in ADOX.
`Column`	Contains information about a specific column from a table, index, or key.
`Columns`	Contains the `Column` definitions of a table, index, or key.

Continued

Table 11-2 *(continued)*

Object	*Description*
`Group`	Contains information about a security group.
`Groups`	Contains all of the security `Group` objects in the catalog.
`Index`	Contains information about an index for a database table.
`Indexes`	Contains the set of indexes associated with a table.
`Key`	Contains the definition of a key field from a table.
`Keys`	Contains the set of `Key` objects associated with a table.
`Procedure`	Contains information related to a stored procedure.
`Procedures`	Contains the set of stored procedures.
`Table`	Contains the definition of a table, including things like columns, indexes, and keys.
`Tables`	Contains the set of tables in the `Catalog`.
`User`	Contains security information about a user account and its database permissions.
`Users`	Contains the set of users associated with the `Catalog`.
`View`	Contains the schema definition of a database view.
`Views`	Contains the collection of `View` objects in the `Catalog`.

The `Groups` collection contains the set of security groups. Each `Group` object contains a reference to the collection of `Users` that are in that particular group. Before you can add a user to the `Users` collection, it must already exist in the `Users` group that is directly beneath the `Catalog` object.

The `Users` collection is similar to the `Groups` collection, except that the users associated with the database are stored in the `Users` collection, while the groups associated with a single `User` are stored in the `Groups` collection. As you might expect, in order to add a group to the `Groups` collection, you must first define it in the `Groups` object that is directly beneath the `Catalog` object.

The `Procedures` collection documents the set of stored procedures available in the database. A single `Procedure` object contains all of the information associated with a specific procedure. It also includes a reference to a `Command` object that can be used to execute the stored procedures.

The Views collection documents the set of views in the database. Each view is defined in the View object. Like the Procedure object, a reference to a Command object that will retrieve the values from the view is also included.

Basic ADO Programming

Writing a database program using the ADO object model is more work than using the ADO Data Control or the Data Environment Designer, but it is far more flexible in the long run.

Connecting to the data source

The first step in building an ADO based application is connecting to the database. The key to this step is creating a connection string that includes key information, such as the data provider, the location of the data, and your user name and password. In addition, you may need to provide additional information, such as the default database name that is required for a specific data provider.

The Connection object uses the connection string to create a way for your program to communicate to the database. Once opened, all database functions performed by your application must reference the Connection object, either explicitly or implicitly.

In addition to providing a gateway to a data source, the Connection object also serves as a common place to record error information by using the Errors collection. The Errors collection contains information about the most recent error encountered. If a new error occurs, the Errors collection will be cleared before any new information is added. It's important to note that a successful operation will not clear the Errors collection. Thus, the Errors collection may not apply to the most recently executed operation. The only way to be certain is to clear the Errors collection after handling the error or immediately before performing a critical operation.

The Connection object is discussed in detail in Chapter 12.

Executing a command

Once you have a connection to the data source, you may issue commands to perform database operations. The commands you issue are defined in a Command object, using the CommandText and CommandType properties. There are three basic types of commands: SQL Statements, table names, and stored procedures.

Once a Command is defined and you have an open Connection to a data source, you may execute the Command. The command may use a set of Parameters to control how it works. This information is kept in the Parameters collection. Parameters can contain values that are passed to the data source and values that are returned from the data source. In addition to returning values through a parameter, a Command may also return a Recordset object, containing a set of rows that were retrieved from the data source.

A Recordset will always be returned if you execute an SQL **Select** Statement or specify a table name for CommandText. All other SQL Statements will not return a Recordset. Stored procedure, on the other hand, may or may not return a Recordset depending on how the stored procedure was written.

The Command object is discussed in detail in Chapter 13.

Playing with Recordsets

A Recordset object is the way you access the actual data stored in your database. The Recordset maintains a pointer to the current row that can be manipulated through the various methods available in the object. You can move to the next row or the previous row, using the MoveNext and MovePrevious methods. You can jump to the first row or to the last row with the MoveFirst and MoveList methods. You can save the current record pointer and later restore it using the Bookmark property.

Access to the information in a particular row is through the Fields collection. Each individual column in the Recordset is represented by a Field object. You can access the current value of the column though the Value property. If the Recordset was opened with update access, you can change the value of the column by assigning a value to the Value property. You can then save the changes to your database by using the Update method or discard the changes by using the CancelUpdate method.

Other functions are available to help you locate a particular row in the Recordset using the Find method , or you can filter the rows currently available in the Recordset to eliminate unwanted rows with the Filter property. You can also refresh the data in the Recordset by using the Requery or Resync method.

The Recordset object is discussed in detail in Chapters 14, 15, and 16.

Thoughts on ADO

Data bound controls are only really useful if you have a visible object. Some types of Visual Basic programs, such as an IIS Application or a COM+ transaction, don't have a visual component, so you can't use bound controls. In general, I find the ADO object model much easier to use than either the DAO or RDO object models. All you need to do is establish a connection to the database and open a recordset and you're ready to go.

When ADO was first released, it didn't contain an object-oriented way to create schemas and for some reason, the Microsoft Jet database (the core database used in Access and frequently used by Visual Basic programmers), didn't include the SQL Statements needed to define schemas. The only way to build a Jet database was to either use the DAO objects provided for schema definition or use a utility program such as the Visual Data Manager or Access to create an empty database. Because Microsoft wanted current DAO users to move up to ADO, they created the ADOX objects. Now there isn't any reason for someone to continue using DAO.

Summary

In this chapter you learned the following:

- You can use the ADO `Connection` object to access a database.
- You can use the ADO `Command` object to run SQL statements and stored procedures on a database server.
- You can use the ADO `Recordset` object to retrieve information from a database and update the information stored in the database.

✦ ✦ ✦