CHAPTER 9

Programming with Data Environments

In This Chapter

The Data Environment Designer

Connecting to databases

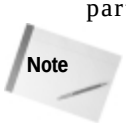
Creating and using commands

Using the Data View window

In this chapter, I'm going to discuss how to use the Data Environment Designer to create a Data Environment, which simplifies many of the tasks of building a database application. It exploits the ADO object model and helps you create standardized ADO objects that simplify access to the database.

The Data Environment Designer

The Data Environment Designer is a tool in the Visual Basic IDE that helps a programmer create a database application more quickly than if they had to perform the same tasks manually. The Designer creates ADO objects corresponding to the tables in your database; it also provides other objects to help you manipulate your database. These objects are available as part of the Data Environment object at runtime.



ADO, not DAO and RDO: The Data Environment Designer only works with the ADO object model. The DAO and RDO object models are not supported. The UserConnection designer is a subset of the Data Environment Designer and supports the RDO object model.

See Chapter 6 for explanations of the ADO, DAO, and RDO models.

Enabling the Data Environment Designer

To add the Data Environment Designer to your project, choose Project➪Add Data Environment from the main menu. If you don't see it listed directly under the Project menu, try looking under the More ActiveX Designers menu (or Project➪More ActiveX Designers➪Data Environment from the main menu). This will display the main window for the Data Environment Designer.

If the Data Environment Designer doesn't appear in either location, you'll need to add it to Visual Basic. Choose Project➪Components from the main menu to display the Components dialog box, then select the Designer tab to see the window as shown in Figure 9-1. Place a check mark in the box next to Data Environment and press OK. This will add the Data Environment Designer to the Project menu. If you don't see Data Environment listed on the Designers tab, you need to install the Data Environment feature from your Visual Basic CD-ROM.

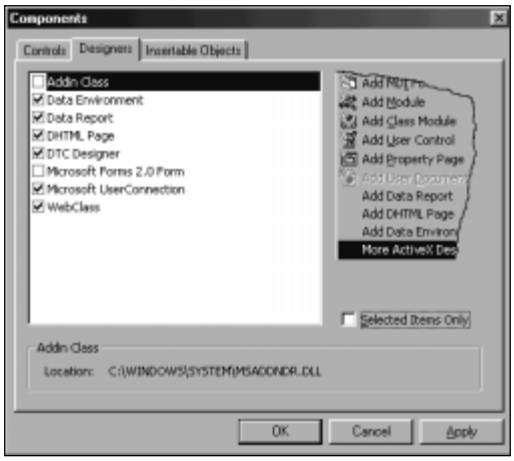


Figure 9-1: Adding the Data Environment Designer to the Visual Basic IDE

Exploring the Data Environment Designer

The main Data Environment Designer window (see Figure 9-2) includes a series of icons that are used to perform common operations, plus an icon tree view of the objects used in the Data Environment.

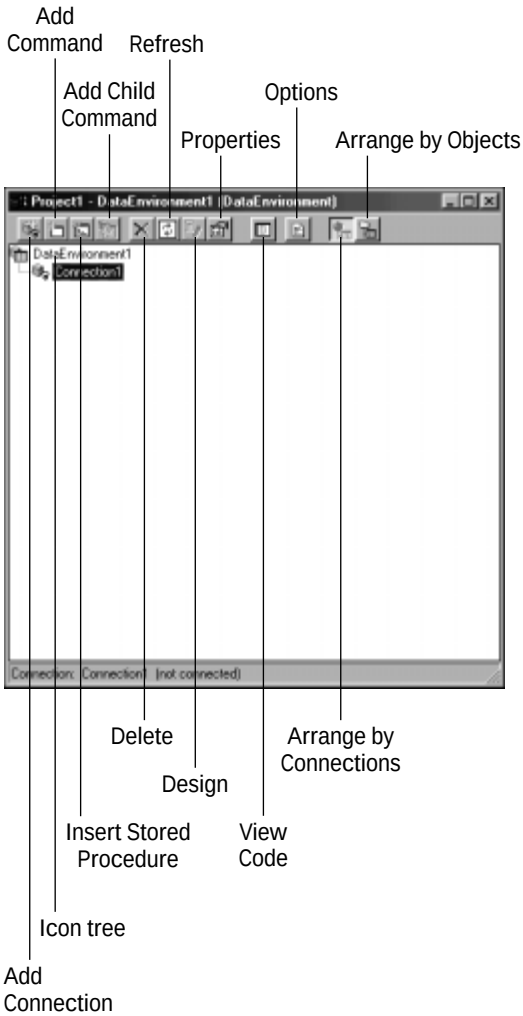


Figure 9-2: Exploring the Data Environment Designer window

- **Add Connection** – adds a new `Connection` object to the data environment, which will be used to access a database.
- **Add Command** – adds a new `Command` object to the data environment, which will be used to execute an SQL command.
- **Insert Stored Procedure** – adds a `Command` object with the information to call a stored procedure on the database server.

- **Add Child Command** – adds a `Command` object to another `Command` object to create a hierarchical structure of data.
- **Delete** – removes an object from the data environment.
- **Refresh** – refreshes the selected object. `Connection` objects are closed and reopened, while all of the metadata associated with a `Command` object will be rebuilt.
- **Design** – accesses the SQL query designer when you specify an SQL statement as the data source for a `Command` object.
- **Properties** – displays the properties associated with the selected object.
- **View Code** – displays the code associated with a particular object.
- **Options** – displays the Options dialog box that controls how the Data Environment Designer works.
- **Arrange by Connections** – arranges the objects in the icon tree by the `Connection` object they access.
- **Arrange by Objects** – arranges the objects in the icon tree by object type.
- **Icon Tree** – the list of objects associated with the data environment.

Many of these functions are also available via a pop-up menu that you can display by right clicking on an object in the icon tree.

Data Environment building blocks

The Data Environment Designer helps you create three main types of objects: `Connections`, `Commands`, and `Recordsets`. These objects are used at both design time and runtime to access your database. At design time, information about your database is extracted and made available to help you create your program. At runtime, these objects are used to perform typical database operations.

It's not what it seems: While these objects appear to be part of the Data Environment Designer, they are really normal ADO objects that the Data Environment Designer helps you create. For more information about all of the capabilities of these objects, please refer to Chapter 11.

Connection objects

A `Connection` object contains the properties and methods needed to access a database. Since the `Connection` object is used at both runtime and design time, a related object called the `DECconnection` is used by the Data Environment Designer to track information about each mode. This information contains the user name, password, how to prompt for password information, and whether to save password information at the end of the session for both runtime and design-time use.

Command objects

A Command object contains the information necessary to execute a command against your database. A command may be an SQL statement, a call to a stored procedure, or the name of a table. Some commands will require you to include a list of parameters when they are run. Some commands will return a set of rows, while others will return nothing but a status condition.

Recordset objects

Rows are returned from your database in a Recordset object. The Recordset object presents a single row of data, known as the *current record*. A *cursor* is used to point to the current record in the Recordset. You can manipulate the cursor by using various methods in the Recordset object.

If the rows in the Recordset object can be updated, there are other methods that allow you to add a new row, delete a row, and update a row. If you need to update multiple rows, you can easily create a Command object, which will call the appropriate SQL statement, or you can use the cursor to move through the rows in the Recordset object one at a time and use the appropriate methods to perform your updates.

Like the ADO Data Control, a Recordset object can be used as a data source with bound controls. This means that you can easily replace the ADO Data Control with the appropriate Recordset object by changing the DataSource property on each of the bound controls.

Hierarchical Recordsets

A hierarchical recordset is based on a normal Recordset, except that in place of one of the values in a row of data is another Recordset, known as a *child* recordset. The child recordset is created by a child Command, whose values are determined by a relationship with the parent recordset. It is also reasonable for a child recordset to have its own child recordsets, thus creating a multi-level hierarchy.

Hierarchical recordsets are useful when you want to retrieve data from multiple tables, but don't want to merge the data into a single table. The classic case of a hierarchical recordset is the customers, orders, and order items. A company has a collection of customers, each customer may have placed multiple orders, and each order a customer has placed may have multiple items.

Connecting to Your Database

The first step in working with the Data Environment Designer is to define a connection to the database. By default the Data Environment Designer creates a single Connection object called Connection1. This object is used at both design time and runtime to access information from the database.

Setting Connection properties

Selecting the `Connection1` object and pressing the Properties button displays the Data Link Properties dialog box, as shown in Figure 9-3. While there are four tabs on the dialog box, only the Provider and Connection tab are typically used. The Advanced tab contains information regarding network settings, connection timeout, and access permissions whose default settings are almost always okay. The All tab lists all of the initialization parameters that will be used when establishing the database connection.

Figure 9-3: Setting Data Link properties for a Connection object

Selecting an OLE DB provider

The first step in defining your connection to the database is to choose an OLE DB provider on the Provider tab of the Data Link Properties dialog box. All of the providers available for your system will be listed. You should choose the native OLE DB provider for your database server, if one is available. Otherwise, you should choose the Microsoft OLE DB Provider for ODBC Drivers. Once you select the proper OLE DB provider, press Next to continue defining the connection properties.

Entering connection information

Since I chose the Microsoft OLE DB Provider for SQL Server in the previous step, pressing Next will request the information shown in Figure 9-4 using the Connection tab. In this case, my database server is on Athena. I'm going to use Windows NT integrated security and I want to use the `VB6DB` database as my default database.

Once I enter this information, I can press the Test Connection button to verify that everything is correct. If there is a problem, a message box will be displayed with a description of the error; otherwise, a message box saying Test connection succeeded will be displayed. After pressing OK on the test's message box, press OK to save the connection information.

Figure 9-4: Entering connection information for the Microsoft OLE DB Provider for SQL Server

Cross-Reference

The information entered in the Data Link dialog boxes is used to build a *connection string*. This value is used anytime you want to connect an ADO object to a database. For more information about connection strings, see the `ConnectionString` property and the `Connection` object in Chapter 11. For details about how to connect to a specific database, see Connecting to SQL Server in Chapter 23, Connecting to Oracle8*i* in Chapter 26, or Connecting to Jet in Chapter 29.

Creating Commands with the Designer

After you have defined a `Connection` object, you can begin to define some `Command` objects. The Data Environment Designer treats `Command` objects in two different ways — as commands and as stored procedures. The difference is primarily how the information for the objects is obtained. If you choose to add a stored procedure, the designer will get a list of stored procedures from the database, you will need to enter a single, dynamically executed SQL statement as one of the properties of the `Command` object. Stored procedures are discussed later in this chapter under "Selecting a Database Object."

Adding a command

Adding a `Command` object by pressing the Add Command button on the Data Environment Designer toolbar simply adds a new command object beneath the currently selected `Connection`. All of the properties associated with the `Command` object are left at their default values and must be changed in order to use the command.

Setting general command properties

Pressing the Properties button or right clicking on the `Command` object will display the `Command1` Properties dialog box as shown in Figure 9-5. On the General tab, you can specify the name of the command, which connection will be used by the command, and the Source of Data for the command.

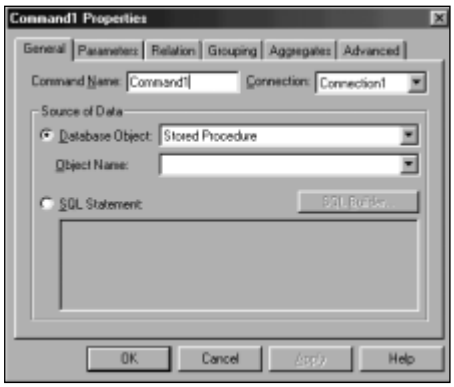


Figure 9-5: Viewing the general properties of the Command1 object

The Source of Data section is somewhat misleading since a `Command` object need not return data. It really depends on whether the database object you select or the SQL statement you enter returns any data.

Selecting a database object

You have a choice of four different database objects: a Stored Procedure, a Table, a View, or a Synonym. A *stored procedure* is nothing but a series of one or more SQL statements that are executed on the database server. Any data returned depends on whether the stored procedure includes a **Select** statement. Selecting Table or View will always return all of the data visible in the selected table or view. A *synonym* is an alternate name for a database object, such as a table or view.

Once you select which type of database object you want, you must specify the object name. The designer makes this easy because after you specify the type of database object you want, it will automatically connect to the database server using

the specified `Connection` object and retrieve the list of available objects. These objects are then made available in the Object Name drop-down list directly below the Database Object drop down box. All you need to do is select the one you want.

Entering an SQL Statement

If you want the command to execute an SQL statement, simply enter the statement in the SQL Statement area in the dialog box. No other statement, except for the **Select** statement, will return data. To make it easier to create an SQL **Select** statement, you can press the SQL Builder button to display the Design window, which will help you build SQL queries.

One of the side effects of using an SQL statement or a stored procedure is that the designer doesn't know what fields will be returned. The only way for the `Command` object to know what fields are returned is to execute the command. Before executing the command, the designer will ask you for permission (see Figure 9-6), since the command could potentially update your database.

Figure 9-6: Asking for permission to execute an SQL statement

Test it first: You should take the time to verify that the SQL statement will work before you enter it into the SQL Statement area. The easiest way to do this is to use a tool like SQL Server's Query Analyzer.

Setting parameters

If you are using a stored procedure as your command, you will probably have one or more parameters associated with it. Consider the stored procedure in Listing 9-1. It will retrieve all of the information about the customer specified in `CustId`.

Listing 9-1: The GetCustomer stored procedure

```
Create Procedure GetCustomer (@CustId Int)
As
Select *
From Customers
Where CustomerId = @CustId
```

Figure 9-7 shows you the Parameters tab of the Properties dialog box. You can view the details about a particular parameter by clicking on it in the Parameters frame. The information related to it will be displayed in the Parameter Properties frame.

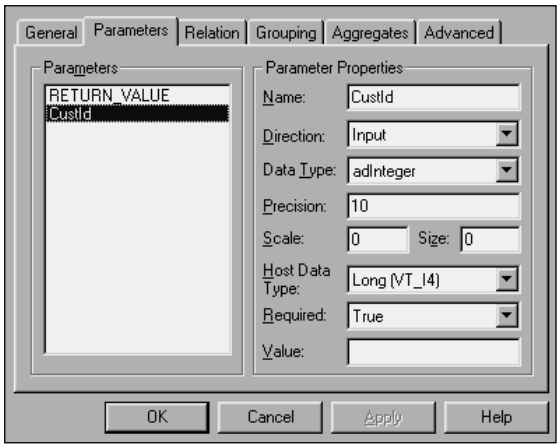


Figure 9-7: Looking at the parameters of a stored procedure

Each parameter has a number of different properties associated with it:

- **Name** – contains the formal name of the parameter.
- **Direction** – classifies the parameter as input, output, or input/output parameter.
- **Data_Type** – specifies the data type associated with the parameter.
- **Precision** – specifies the precision of a numeric data type.
- **Scale** – specifies the scale of a numeric data type. This field is disabled for non-numeric data types.
- **Size** – specifies the size associated with the data type. This field is disabled for non-numeric data types.
- **Host Data Type** – specifies the data type that will be used by the host. This value must be compatible with the value specified in the Data_Type field.
- **Required** – is `TRUE` when you must specify a value for the parameter before the command can be executed.
- **Value** – contains the default value for the parameter. It will be used at design time to test the `Command` object. At runtime, you will typically override this value with the real value you wish to use.

I can't change anything: If the type of command you are creating doesn't allow parameters, all of the fields on this tab will be blank, and even if the type of command does support parameters, some of the properties of the parameter may be disabled, meaning that you can't change them.

Setting advanced properties

The Advanced Properties tab (see Figure 9-8) contains a number of different properties that affect how the command will be executed and how the results will be returned. Depending on the type of command you choose, some or most of these properties may be disabled.

Figure 9-8: Reviewing advanced properties

Reviewing Recordset management properties

The four properties in the *RecordSet Management* section (`CursorType`, `CursorLocation`, `LockType` and `CacheSize`) control how the `Recordset` object will be created and how it can be accessed from your program.

See the `Recordset` object in Chapter 14 for a more complete discussion of cursors and locks.

The type of cursor used in a `Recordset` determines many of its characteristics. Client-side cursors are restricted to using only static cursors, while server-side cursors can use any of these four different types of cursors:

- **Forward Only** cursors point to a collection of rows that can only be accessed in a forward direction. Moving backwards isn't permitted. Otherwise, this cursor type is identical to Static.
- **Keyset** cursors point to a collection of rows that see all changes (including deletions) to the database, except for records that have been added after the `Recordset` was created.

- **Dynamic** cursors point to a collection of rows that see all changes (including additions and deletions) to the database after the `Recordset` was created.
- **Static** cursors point to a collection of rows that will not change until the `Recordset` is closed. Any rows that have been added, deleted, or updated by any other database user are not seen.

The `CursorLocation` property describes where the cursor is located. Choosing Use server-side cursors means that the cursor will be managed by the database management system, while choosing Use client-side cursors means that the cursor will be managed by a cursor library located on the same computer as the application. Client-side cursors are more flexible than server-side cursors, while server-side cursors are easier to use than client-side cursors.

The `LockType` property determines how you and other users can access the data in the database.

- **Read-only** locks mean that you can't alter any of the data in the `Recordset`.
- **Pessimistic** locks mean that the row is locked immediately when you begin to change the values in a row. The lock will be released when you complete your changes.
- **Optimistic** locks mean that the row is not locked until you have finished making your changes. Then the row is locked and the current values of the row in the database are checked against the original values before you changed them. If they are the same, your changes will be saved to the database, otherwise an error will be returned to your program.
- **Batch Optimistic** locks work just like optimistic locks, except that multiple rows of information are updated locally and are sent to the database server in a single batch. Any errors will be returned to your program. Your program must then identify the rows that weren't updated and take the appropriate action for each row.

Lock least: You should lock the least amount of data possible in order to make it easy to share data. If you don't need to update any of the data, you should use a read-only lock. In the beginning, you should use a pessimistic lock with a server-side cursor to prevent others from updating the data you are editing. Once you are comfortable with database programming, you should consider using client-side cursors with either optimistic or batch optimistic locking.

The `CacheSize` property determines how many rows are buffered in the client machine. Increasing this size allows the database server to operate more efficiently when returning data to the client system. It will also improve the performance of the client system, since most of the time, it will retrieve data from the local cache rather than request data from the server. However, the data in the cache will not be updated if changes are made to the same rows on the database server. The more rows you keep in the cache, the greater the chance that someone else may have

updated the data. If you choose pessimistic locking, you should set the `CacheSize` property to one to prevent invalid records from sitting in your cache.

Reviewing other advanced properties

Like the properties in the *RecordSet Management* section, those in the *Command Configuration* section are taken from the `Command` object discussed in Chapter 11. However, the `Command Timeout` is important enough to discuss twice. This property contains the number of seconds the command is allowed to run before it is canceled. When dealing with large volumes of data or very complex stored procedures, the default value of 30 seconds may not be enough to allow the command to finish. If this happens, you will need to increase this value.

The *Call Syntax* section describes how a stored procedure will be called, including the list of parameters and the return value. While you can edit this value to change the number of parameters and presence of a return value, in general you should leave this field alone.

Saving the Command

To save your property settings and continue working with the properties, press the Apply button, or to save your changes and close the Command1 Properties dialog box, press the OK button. If the Apply button is grayed out, it indicates that the saved version of the environment is the same as the one you are viewing. Once your `Command` has been defined, you can see the list of fields that the command will return in the Data Environment by expanding the plus sign in front of the command's icon (see Figure 9-9).

Figure 9-9: Viewing the fields returned by a command

Adding a Child Command

One of the nicer features of the Data Environment Designer is its ability to create hierarchical recordsets easily. Simply select the command you want to add the child to and press the Add Child Command button. This creates a new command that is nested at the same level as the fields on the parent command. Note that the command you choose as the parent command must return a `Recordset`, otherwise you will not be able to add the child command.

After selecting the child command, you can press the Properties button to define its properties. The first thing you must do in the Properties dialog box is to define the Source of Data. As with the parent command, you must choose a Source of Data that returns a `Recordset`. If you don't define a source of data, you'll receive a warning message if you try to access any of the other tabs in the Properties dialog box.

Once you've defined the Source of Data, you should define any properties on the Parameters and Advanced tabs that are needed to retrieve the data. Note that the `Recordset` object returned by the child command is always read-only. In fact, most of the properties on the Advanced tab will be disabled, and the properties will be set to be compatible with the properties selected for the parent command.

Defining a relationship

On the Relation tab (see Figure 9-10), you must define how the parent and child commands are related. At the top of the page, you'll see the Relate to a Parent Command Object check box and the Parent Command drop-down box. By default, the check box will be checked, meaning that this command is a child command, and its parent command will be the one that was selected when you pressed the Add Child Command button. Unchecking the check box will make this a regular command that is independent of any other commands in the Designer, while selecting a different parent command will make this command a child of the selected command.

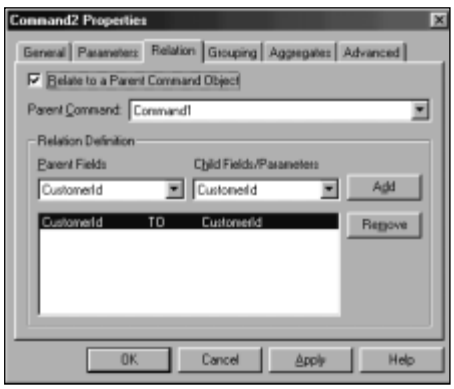


Figure 9-10: Defining a relationship between a child command and its parent

Once the parent has been selected, you need to choose the fields that establish the parent child relationship in the Relation Definition frame. Specifying pairs of fields, one in the parent and one in the child, defines the relationship between the two commands. For each row retrieved by the parent command, the child command will be executed and the values from the parent fields will be used to select only those rows with matching values in the child command. Since this is a somewhat complex concept, let's use an example from the sample database. Assume that the parent command retrieves all of the rows from the Customers table. Then the child command is set up to retrieve all of the rows from the Orders table. The relationship information specified on the Relation tab associates the CustomerId value in the Orders table with the CustomerId value in each row retrieved from the Customers table.

The information needed to build the sample database can be found in the \SampleDB directory on the CD-ROM.

The retrieved records would look like the information in Table 9-1. Notice that for each row retrieved from the Customers table (CustomerId and Name), one or more rows of information will be retrieved from the Orders table (OrderNumber and DateOrdered).

Table 9-1
A View of a Hierarchical Recordset

CustomerId	*Name*	*OrderNumber*	*DateOrdered*
0	Dexter Valentine	-	-
0	-	1000	1 Jan 2000
1	Malik Hubert	-	-
1	-	1001	1 Jan 2000
1	-	1004	1 Feb 2000
2	Lee Holt	-	-
2	-	1002	2 Jan 2000
2	-	1005	2 Feb 2000
3	Scotty Waltrip	-	-
3	-	1003	3 Jan 2000
3	-	1006	3 Feb 2000
3	-	1007	3 Mar 2000

Using groupings

An alternate way to create a hierarchical recordset is to define a grouping using the Grouping tab of the Properties dialog box (see Figure 9-11). A *grouping* allows you to take a single recordset and break it into two levels. A new level is created each time the values of the *Fields Used for Grouping* change.

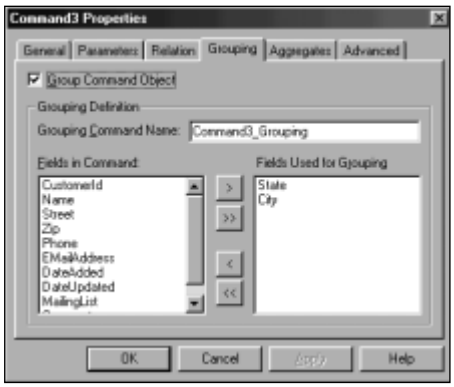


Figure 9-11: Selecting fields for grouping

To define your grouping, simply select the field or fields you want to move in the *Fields in Command* section of the form and press the > button to move them to the *Fields Used for Grouping*. You can move a field in the reverse direction by using the < button. The >> and << buttons move all of the fields in the direction of the arrows.

Aggregating data

In the Aggregates tab of the Command Properties dialog box (see Figure 9-12), you can create summary data for your recordset or hierarchical recordset. You can define aggregates that are based on grouping levels. You can also define grand totals for the entire recordset. The aggregated data will appear as part of a hierarchical recordset.

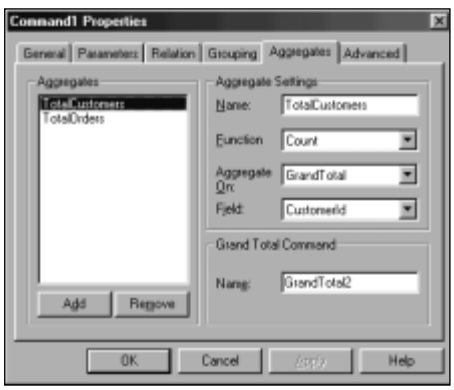


Figure 9-12: Selecting aggregations for a hierarchical recordset

To add a new aggregation, press the Add button to create a default aggregation. A new aggregation will appear in the Aggregation frame. Clicking on the aggregation will display all of its settings in the Aggregation Settings frame. Also, you can remove an aggregation by selecting it in the *Aggregates* frame and pressing the Remove button.

When working with a hierarchical recordset, you can define aggregations in two ways — as a grand total, which is computed over the entire recordset, or as a group summary over a child command. You can't define an aggregation on the lowest level of a hierarchical recordset. If you are working with a regular recordset, you can only define aggregations over the entire recordset for a grand total.

If you aggregate values over a child command, the aggregated values will be added to the current level of the hierarchical recordset. If you create a grand total, it adds a new level above the current level of the hierarchical recordset. When adding an aggregation, you need to give it a name and decide how to compute it. Remember that an aggregation will appear as a field in the recordset, so using a meaningful name is important.

To compute an aggregation, you need to define three pieces of information: the level of aggregation (the name of a child command or grand total), the name of the field that you want to aggregate, and the name of function that you want to use to perform the aggregation (see Table 9-2).

Table 9-2
Aggregation Operations

Operation	***Description***
Any	Performs the default operation for the field.
Average	Averages the values for a particular field.
Count	Counts the number of rows retrieved.
Maximum	Returns the highest value found in the field.
Minimum	Returns the lowest value in the field.
Standard Deviation	Computes the standard deviation of all of the values for the selected field.
Sum	Adds the values of the specified field together.

Inserting a stored procedure

Even though a stored procedure uses the same property window as regular `Command` objects, a special dialog box makes it easy to add multiple stored procedures to the Data Environment Designer. Clicking the Insert Stored Procedures button will display

the dialog box shown in Figure 9-13. Simply select the stored procedures from the Available list and press the > button to add them to the Add list. You can use the >> button to add all of the stored procedures to the Add list. Pressing the < or << buttons will remove the selected or all of the stored procedures from the Add list respectively.

Once you've created the list of stored procedures you want to add, press the Insert button to add them to the Designer. When all of your selected stored procedures have been added, you can repeat the process to select additional stored procedures, or press the Close button to return to the Data Environment Designer.

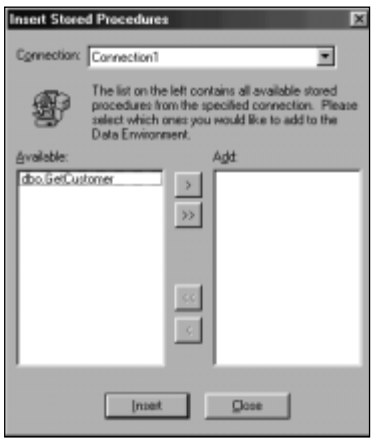


Figure 9-13: Inserting a stored procedure

Building Programs with the Designer

The true power of the Data Environment Designer is shown when you use it to design your forms. The Designer allows you to drag commands and fields from the Designer onto a Visual Basic form to create controls that are bound to the objects in the runtime component of the Data Environment Designer. Once the controls are positioned, you can write code to manipulate the information on the form.

Drawing controls

If you expand a `Command` object, you will see the list of fields in the object (see Figure 9-14). To add a field to your Visual Basic form, simply move the cursor to the field you want to add, press the left mouse button, and drag the field onto the form while still pressing the left mouse button. When the control is where you want it, release the left mouse button, and the control will be added to your form.

The new control is automatically bound to a `Recordset` object that is owned by the runtime component of the Data Environment Designer. The additional fields that are dragged onto the form from the same command will also be automatically bound to the same `Recordset` object. In addition to dragging fields, you can also drag a command onto your form. All of the fields that are contained in the command will be dropped on the form. Figure 9-15 shows the effect of dragging `Command1` from Figure 9-14 onto a blank form.

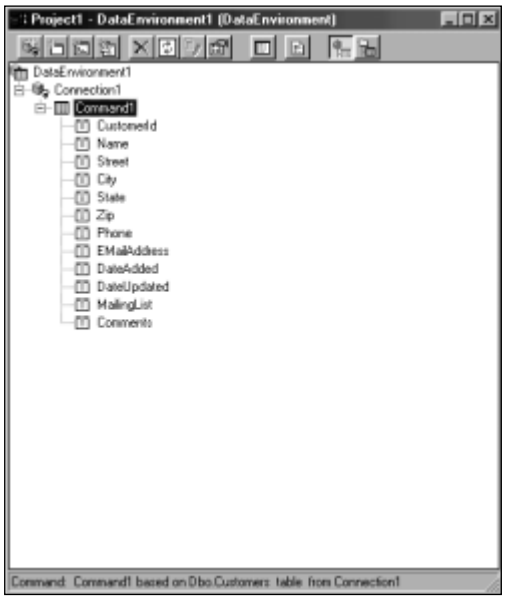


Figure 9-14: Viewing the fields in a command

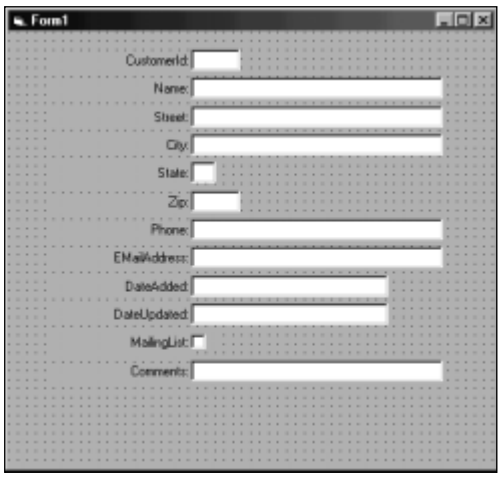


Figure 9-15: The result of dragging a command 8object onto a form

They're still in one piece: When you drop the fields onto your form object, all of the fields are selected. This makes it easy to move the fields around on the form until they fit.

Setting options

You can control how a field appears on a form by setting the Field Mapping on Data Environment Designer's Options dialog box (see Figure 9-16). Depending on the data type of the database field, the Designer will select a control type based on the information shown in the Default Control Association area of the dialog box.

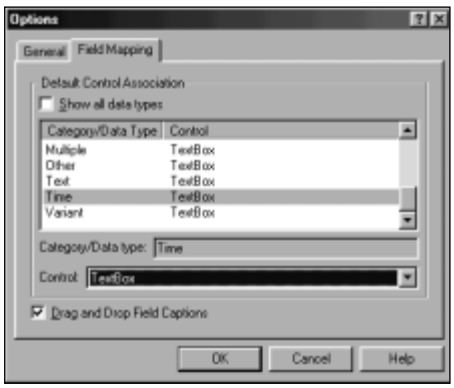


Figure 9-16: Changing the field mapping

If the Drag and Drop Field Captions check box is checked, a `Label` control will automatically be generated alongside the selected control. The `Caption` property of the `Label` control will be set to the name of field in the database.

You can change the control associated with a data type by selecting the data type under *Category/Data Type* in the list box area (see Figure 9-16). The data type will appear in the *Category/Data type* area beneath the list box, and the control will appear in the drop-down box below that. To change the control, simply press the drop-down arrow to display a list of choices. All possible controls on your system will be listed, whether or not they are available in your current project. If you choose a control that isn't available in your project, it will be added automatically. Press OK to save your changes.

There's more: On the General tab of the Options dialog box, there are some general options you can enable or disable. While none of these options have a big impact on how the Designer works, you might find that they make the Designer more comfortable to use.

Data Environment RunTime Object Model

The Data Environment Designer includes a runtime object called `DataEnvironment` for your program. The `DataEnvironment` object is basically just a container for all of the objects you created in the Designer.

DataEnvironment properties

The `DataEnvironment` object contains five main properties, which are listed in Table 9-3. The `Commands` and `Connections` are references to collection objects containing the commands and connections you defined using the Data Environment Designer.

Table 9-3
Properties of the DataEnvironment Object

Property	*Description*
`Commands`	An object reference to a collection object containing the set of `Commands` objects defined in the Data Environment.
`Connections`	An object reference to a collection object containing the set of `Connection` objects defined in the Data Environment.
`Name`	A `String` containing the name of the `DataEnvironment` object.
`Object`	An object reference to the `DataEnvironment` object.
`Recordsets`	An object reference to a collection object containing the set of `Recordset` objects defined in the Data Environment.

The `Recordsets` collection contains a series of `Recordset` objects. Each `Command` in the `Commands` collection will have a corresponding `Recordset` object in the `Recordsets` collection, which will hold the results of the `Command`. Adding the characters `rs` to the front of the `Command` object's name forms the name of the `Recordset` object.

Unlike traditional collection objects, you can't add or delete objects from these collections at runtime. The only way to add or delete objects from these collections is by using the Data Environment Designer at design time.

Using these collections can be a little difficult. The following line of code is used to retrieve the value of the CustomerId column:

```
DataEnvironment1.Recordsets("rsCommand1").Fields("CustomerId").
Value
```

You need to explicitly include the name of the Recordset when trying to access it through the collection object. Since this approach is pretty messy, Microsoft included a shortcut for each Recordset object in the DataEnvironment object. This allows you to rewrite the statement as:

```
DataEnvironment1.rsCommand1.Fields("CustomerId").Value
```

Then if you use the With statement, you can rewrite this statement as:

```
With DataEnvironment1.rsCommand1
   .Fields("CustomerId").Value

End With
```

With what: Using the With statement can be very useful if you need to perform multiple operations against the Recordset. Not only do you save yourself a lot of typing, your program will execute slightly faster. The better performance is a result of having to resolve the reference to the DataEnvironent1.rsCommand1 object once for the With block rather than once for each statement executed.

DataEnvironment methods

The default DataEnvironment object has no default methods. However, each time you add a command in the Data Environment Designer, a shortcut method will be added to the DataEnvironment object. This makes it easier to code your application.

Calling a command without parameters

There are two ways to call a Command object using the DataEnvironment object. You can access the command though the Commands property, like this:

```
DataEnvironment1.Commands("Command1").Open
```

Or you can use the shortcut method, like this:

```
DataEnvironment1.Commmand1
```

Calling a command with parameters

It isn't difficult to call a Command object with parameters. You can specify the parameters as part of the Command object's Parameters collection, as shown below:

```
DataEnvironment1.Commands("Command1").Parameters("Parm1").Value = 1
DataEnvironemnt1.Commands("Command1").Parameters("Parm2").Value = 2
```

```
DataEnvironment1.Commands("Command1").Parameters("Parm3").Value = 3
DataEnvironment1.Commands("Command1").Open
```

Or you can use the shortcut method, like this:

```
DataEnvironment1.Commmand1 1, 2, 3
```

See Chapter 11 for more information about the Parameters collection.

Data Environment events

The Data Environment Designer automatically creates a separate module to hold all of the code for the `DataEnvironment` object and all of the `Connection` and `Recordset` objects created beneath it.

Event Initialize ()

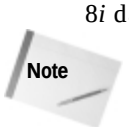
The `Initialize` event is called when the `DataEnvironment` object is accessed for the first time. Typically, this will be when your program is first started. This is a good place to include code that prompts the user for a user Id and password if you don't want to use the default login form.

Event Terminate ()

The `Terminate` event is called just before the `DataEnvironment` object is destroyed.

Viewing Databases with the Data View Window

The Data View Window is the central component of the Visual Database Tools (see Figure 9-17). It works with the Data Environment Designer to allow you to perform various tasks with your database, such as designing your database, editing data in the tables, creating **Views,** and managing stored procedures. These tools are based on the tools that ship with SQL Server 7, and work with both SQL Server and Oracle 8*i* database systems.



Why doesn't it work for me?: Some of the features of the Data View Window are only available with the Enterprise Edition of Visual Basic.

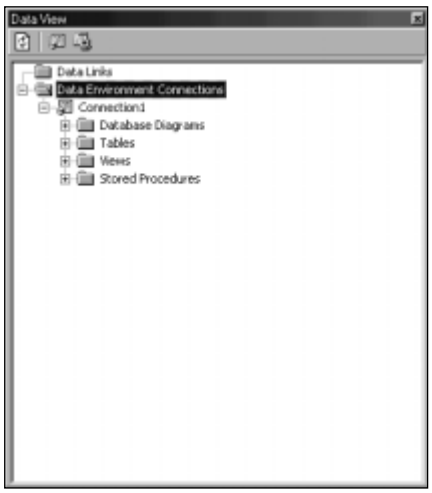


Figure 9-17: Viewing databases with the Data View Window

Configuring the Data View Window

To start the Data View Window, you can choose View➪Data View Window from the main menu or press the Data View Window icon on the toolbar. You can also display the Data View Window by pressing the Design button while in the Data Environment Designer.

All of the `Connection` objects you defined in the Data Environment Designer will be listed under the Data Environment Connections icon. Simply expand the icon to show icons for Database Diagrams, Tables, Views, and Stored Procedures. Expanding these icons will show you the list of database objects you can access through the Designer.

Data Environment Not: You don't need to use the Data Environment Designer with the Data View Window. You can create a Data Link by right clicking on the Data Links icon and selecting Add a Data Link from the pop-up menu. You'll see the familiar Data Link Properties dialog box that you've seen many times before. Simply select the OLE DB provider for your database and press the Next button. Then enter the rest of the information and you'll be all set to use the Data View Window.

Working with database diagrams

A *database diagram* is a visual representation of a database that you can modify (see Figure 9-18). You can add new tables, delete existing tables, or change any of the characteristics of a table.

The Database Diagram tool is identical to the Database Diagram tool in SQL Server 7's Enterprise Manager. See Chapter 24 for more information.

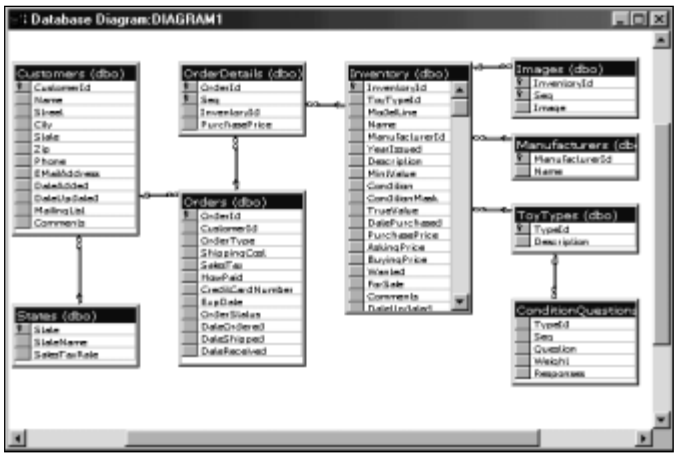
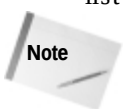


Figure 9-18: Working with a database diagram

Working with tables

There are two main functions you can perform with tables using the Data View Window. The first is to open a table in design mode (see Figure 9-19) by simply right clicking on the table name and selecting Design from the pop-up menu.

Each column in the table corresponds to a row of information displayed in the Design window, including the Column Name, its Data Type and size information, whether **Nulls** are allowed, etc. . . You can easily change most values directly by editing the value in the particular cell. For instance, you can changethe length of a **Varchar** field from 50 characters to 128 characters by changing the value 50 to 128. You can also add new columns to the table by simply adding them to the end of the list of columns.



But it's not empty: Even if your table has data in it, you can still make some changes to its structure in Design mode. Before the changes are applied to the database, the utility will verify that the data in the table is compatible with the changes. If it isn't, an error message will be displayed and you can opt to save your changes into a script file that can be applied later.

The other main function you can perform with a table is to view its data in a spreadsheet-like grid, similar to the `DBGrid` control (see Figure 9-20). You can open your table by right clicking on the table name and selecting Open from the pop-up menu.

You can change any value in the table by simply overtyping the data in the particular cell. The changes are committed to the database when you move your cursor to another row. If you move your cursor to the first blank row at the end of the grid, you can insert a new row into the table by right clicking on the table and choosing New from the pop-up menu.

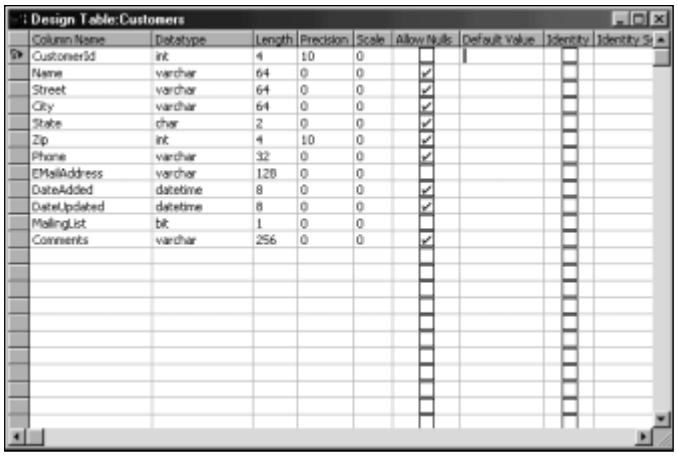


Figure 9-19: Working with a table in design mode

Figure 9-20: Opening a table for viewing

You may select rows by clicking on the row header area, just before the first column. Pressing the delete key will display a dialog box asking if you really want to delete the selected rows from your database. Pressing Yes will remove the rows, while pressing No will return you back to the table.

Working with views

The same two main functions available with tables are also available for views. You can open a view just like you opened a table, and the results will be displayed the same way. You can also design a view using the Query Designer (see Figure 9-21).

You can enter your **Select** directly or use the interactive Designer to create the **Select** statement.

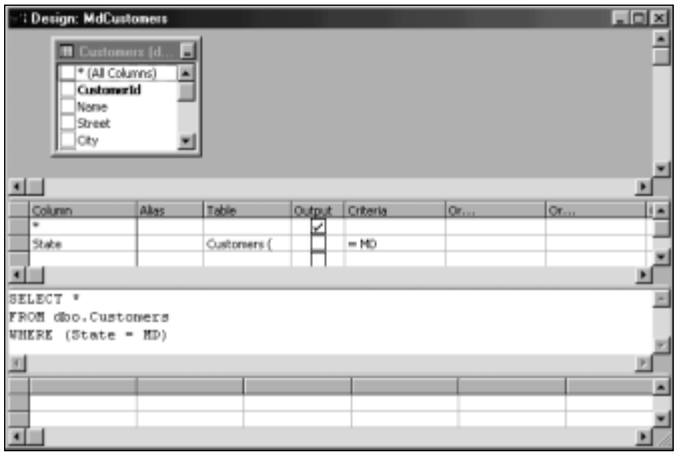


Figure 9-21: Designing a view

Working with stored procedures

You can design and debug stored procedures using the Data View Window. Right click on the stored procedure you want to edit and choose Design from the pop-up menu. This will display the Design window for the stored procedure (see Figure 9-22). You can type your changes into the window and save them as a disk file or back to the database. When you are finished, you can use the T-SQL Debugger to test it.

Figure 9-22: Changing a stored procedure

Thoughts on the Data Environment Designer

The Data Environment Designer is an alternate way to develop database applications. It is a step up in functionality from the ADO Data Control that was discussed in Chapter 7, but it is not as flexible as using the ADO objects, directly. The primary advantage to using the Data Environment Designer is that it allows you to drag and drop bound fields to your form objects. It is also useful when you want to create hierarchical recordsets for use with Microsoft Report, which I'll cover in Chapter 10.

The Data View Window allows Microsoft to reuse some of the code in SQL Server to provide some of the basic design facilities that exist in SQL Server's Enterprise Manager in Visual Basic. This means that you don't have to have the SQL Server software installed on your development computer. While this is an interesting tool in theory, I find myself using SQL Server's Enterprise Manager or Oracle's SQL*Plus for these functions anyway. I invariably need Enterprise Manager or SQL*Plus open to perform a function that isn't available in the Visual Database Tools or to look at a piece of data while I'm running my program, because the Data View Window is available only at design time.

Summary

In this chapter you learned the following:

- You can use the Data Environment Designer to simplify the process of creating various ADO objects that are used to access your database.
- You can create `Command` objects to execute SQL statements.
- You can explicitly specify an SQL statement for a `Command` object or create a reference to a table or stored procedure.
- You can use the Data View Window to view the contents and structure of a database.
- You can design SQL statements and create stored procedures directly from Visual Basic.

✦ ✦ ✦