Click & Retrieve Source CODE!

Mapping Objects to Databases

Use classes to map objects to databases and enhance your object-oriented development environment.

by Ken Fitzpatrick

Now that VB4 offers the new class module and Collection object, Visual Basic developers can begin to use real object-oriented design and programming techniques while building their applications. You can not only take advantage of OLE Automation, but also take VB programming to a higher level by developing powerful applications based on robust Business Object Models (BOMs). Developing a nonvisual BOM first, instead of taking the conventional, GUI-centric approach to application design, allows you to make frequent, even drastic, changes to the model of your application without changing the corresponding GUI and database components (for more information, see the accompanying sidebar, "Don't Drop the BOM").

But how far can you take object-oriented development in VB4, where the class module and the Collection object are essentially the only available object-oriented programming tools? How do these tools fit in with the other great aspects of VB, such as its language, graphical user interface, and database features? How does the VB database factor into the object-oriented world?

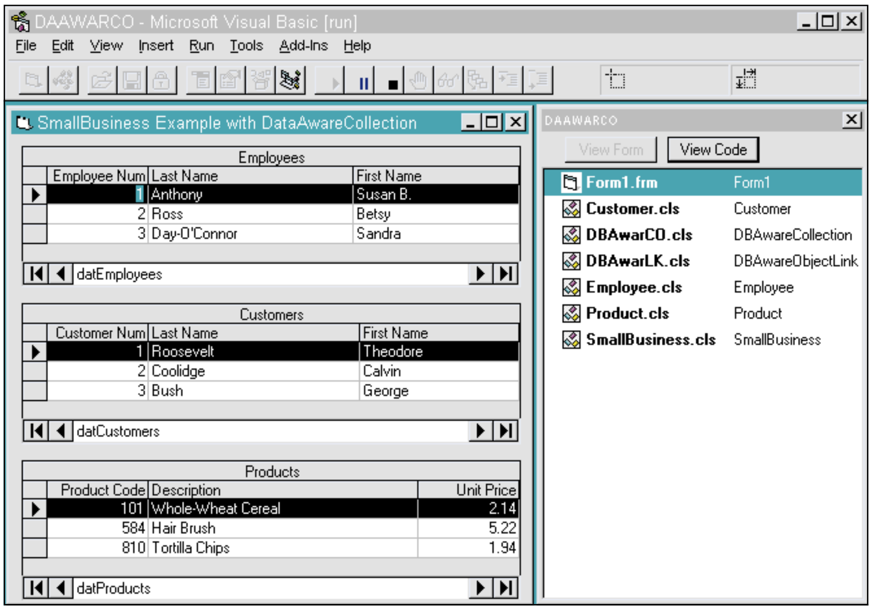
Wouldn't it be nice if all of these VB features were seamlessly integrated with one another to create an enriched VB application development environment? Wouldn't it be great to be able to use the same object-oriented techniques of managing class modules and Collections as the primary means of managing the other VB features such as the RecordSet, ListBox, ComboBox, DBGrid, and other bound controls? Using the same approach to develop both the object model and user interface code would significantly reduce your development time.

In addition to working as a Visual Basic and SmallTalk programmer/analyst at USAA in San Antonio, Texas, Ken develops shareware products that enhance Visual Basic's object-oriented features. Reach Ken by e-mail at 73367.3470@compuserve.com or KenFitzpatrick@msn.com.

Even though this level of object-oriented capability isn't readily available in VB4, the basic tools to provide these features are available. It just takes some thought and crafty coding. One way to apply object-oriented techniques to your application is to directly associate your application's objects with its database. You may be familiar with the object-oriented concept of encapsulation: objects "know" certain details about how they will be used. Mapping objects to a database means that the object must have knowledge about how its properties relate to database fields. With this knowledge centralized, you can change either the object or the database without affecting any outside component, and your application doesn't need to be concerned with the details of the mapping exercise each time it accesses the database. I'll show you how to map objects to databases, then I'll describe how to add data-management support to your classes by replacing the VB4 Collection object with an equivalent object that supports data-aware behavior.

THE LONELY CLASS MODULE

For the several years since the release of Visual Basic 2.0, the "VB Threesome"—its highly integrated language, GUI, and database capabilities—has provided an excellent platform for developing rich, GUI-based applications. But VB4's new object-oriented features, namely its class module and Collection object, are currently isolated and excluded from the


DAAWARCO - Microsoft Visual Basic [run]
File Edit View Insert Run Tools Add-Ins Help
SmallBusiness Example with DataAwareCollection
DAAWARCO
View Form
View Code

Employees

Employee Num	Last Name	First Name
1	Anthony	Susan B.
2	Ross	Betsy
3	Day-O'Connor	Sandra

datEmployees

Customers

Customer Num	Last Name	First Name
1	Roosevelt	Theodore
2	Coolidge	Calvin
3	Bush	George

datCustomers

Products

Product Code	Description	Unit Price
101	Whole-Wheat Cereal	2.14
584	Hair Brush	5.22
810	Tortilla Chips	1.94

datProducts

Form1.frm	Form1
Customer.cls	Customer
DBAwarCO.cls	DBAwareCollection
DBAwarLK.cls	DBAwareObjectLink
Employee.cls	Employee
Product.cls	Product
SmallBusiness.cls	SmallBusiness

FIGURE 1 ***An Object for an Object.*** *You can easily transform the SmallBusiness example, found in Chapter 7 of the* VB Programmer's Guide, *into a database application by replacing its Collection objects with data-aware Collection objects. This illustration shows how the data-aware Collection object can control the data control's RecordSet property.*

threesome. In order to fully exploit these features, you must integrate the class module and Collection object into the mainstream VB development environment. Arguably, the easiest way to integrate them is to integrate them into any one of the three components, most likely the GUI or the database. Integrating into both would be even better.

Because the solution should support the nonvisual BOM approach, you shouldn't link the class modules to the GUI as the initial target. This leaves the VB4 database as the initial point of integration of the VB4 class module into the threesome.

In order to integrate the class module and Collection object into the VB database, you must resolve these problems:

- How do you map a class module to a data source?
- How does a row in the data source map to an object?
- How can you implement the BOM object containment hierarchy over a relational database model?
- How can you support the needs of the VB GUI?
- How can you process VB GUI controls in an object-oriented manner?

It seemed reasonable that in addition to the class module, the VB4 Collection object should play a key role in the integration process. However, for successful integration, the role of the Collection object should automatically and implicitly interface with the VB database.

Because the VB4 Collection object doesn't have methods that allow it to access databases, you would need to either enhance it or replace it with an equivalent object that does have methods for accessing databases. Because enhancement wasn't an immediate option, I elected to functionally replace the Collection object with an equivalent object that adds automatic data-management capabilities to its object-oriented functions. In other words, I supplemented the Collection object's methods such as Item, Add, and Remove with several others geared to supporting the data-aware capabilities of the replacement Collection object.

THE OBJECT MUST HAVE KNOWLEDGE ABOUT HOW ITS PROPERTIES RELATE TO DATABASE FIELDS.

But before discussing the replacement for the Collection object, let me start with the first problem I listed: mapping the class module to the data source. When you're working with a homogeneous development environment consisting of an object-oriented language and an object-oriented database, this is easy. It's natural for object-oriented languages such as SmallTalk to store their objects in an object-oriented DBMS. But you're faced with a heterogeneous environment when you work with VB4: VB4's class modules have object-oriented capabilities, but they'll be connecting to a relational database such as Microsoft Jet. Because the VB application must store the data for its objects in a relational DBMS, you'll have to do a little work to provide a correlation between the DBMS and the class modules.

One of the first steps in resolving this issue is to decide the appropriate place to store the name of the applicable data source for each class module. It is considered an acceptable object-oriented practice for an object to be aware of certain

Don't Drop the BOM

One of the most interesting opportunities presented by object-oriented technology is that it allows you to develop a nonvisual business object model (BOM) before applying any GUI or database features to the application. In the same way you should lay and verify the foundation of a building before the walls go up, you should develop and test a nonvisual BOM before you finalize the user interface and databases.

The BOM-first approach allows you to model frequent, even drastic, changes without having to perform simultaneous maintenance to corresponding GUI and database components. In fact, the longer you can defer the design and finalization of the user interface and databases, the better, because you'll be able to remain flexible during the early stages of development. This technique supports wide-open creativity because it encourages the pursuit of numerous, varying, and unconventional design approaches that can be modeled and evaluated cheaply and quickly. This would not be considered an option under conventional, GUI-centric design techniques because of the time and costs associated with taking such ventures.

Rather than following the conventional "waterfall" methodologies to application design and development, this approach supports cyclical development: each subsequent iteration of the BOM typically offers more than the previous and is closer to being "correct." Nearly all first-time object-oriented development projects go through several "false-start" scenarios, but the very nature of object-oriented design prevents any of these from actually being considered a loss or a failure. Instead, each subsequent iteration is typically an improvement on the previous one, and each contributes something to the ultimate solution.

Another advantage of the nonvisual BOM is that you can deploy it in many forms, not just under a GUI. For example, class modules that have no user interface are ideal for deployment as OLE components to support reports, spreadsheets, and other OLE Automation purposes. By comparison, if the class modules contain user interface code or if the business rules are interlaced throughout the GUI, as is the case with conventional VB development approaches, it might be difficult to provide such a multifaceted degree of support.

A data-aware replacement for the Collection object integrates well with the BOM-first approach, because its capabilities include effortlessly bringing RecordSet management capabilities to the role of the collection. This allows you to design and develop the BOM as usual before any database is available or even defined. Then, when the database becomes available, you can exploit it through the data-aware collection without any changes to the BOM. While you're developing the BOM, you can focus on meeting the business requirements, rather than making allowances for future RecordSet processing.—*K.F.*

details of how it will be used, such as this, and to provide that information to any code that calls the object through one of its public methods.

For purposes of this article, the class module needs to know only the name of the data source to which it will map. However, depending on any number of factors, it might be appropriate for the class module to know other database-mapping processes, such as a SQL statement that retrieves data rows for itself or its contained objects. It's best to resolve this issue on a case-by-case basis, according to your application's requirements. In general, it's a good idea to have a one-to-one correspondence of class modules to database tables or updateable queries. Each class module, then, would equate to its applicable data source.

Each class module should have a public method that returns the name of the applicable data source (or other information). In the interest of polymorphism, you should decide on a common method name such as "TableName" or "DataSourceName":

```
Public Function DataSourceName() _
   As String)
' Return the name of my
' associated data source
   DataSourceName = "Employee"
End Sub
```

If the class module is the proper place to encapsulate information about its associated data source, then it must also be the proper place to encapsulate details about mapping a row of the data source to its internal properties. Each class module should have a public method that copies values from a row of the referenced data source to the class module's properties. The data-aware Collection-object replacement would execute this method any time it has a RecordSet row that needs to be translated into an object. Again, decide on a polymorphic method name such as "InitializeFromRecordSet." Here's an example of the program code found in this method:

```
Public Sub InitializeFromRecordSet_
   (RecordSet As RecordSet)
   MyLastName  = RecordSet("LastName")
   MyFirstName = _
      RecordSet("FirstName")
   ObjectID = RecordSet("ObjectID")
End Sub
```

Note that this example includes a reference to the property "ObjectID." You might need a property of this type to identify the object within the Collection object and, likewise, the row within the corresponding data source. In general, a good candidate for this field in the database table is the surrogate-key column, which has the Counter attribute and is mapped to a class module property that has the Long attribute. It's a good idea for all class modules and database tables to use an identical field name for this purpose.

Each class module should have a public method, such as "InitializeRecordSet," that copies values from the class module's properties to the data source row (the reverse functionality of the previous method). The data-aware Collection-object replacement would execute this method any time it needs to update the corresponding RecordSet row with the current property values on the object:

```
Public Sub _
   InitializeRecordSet_
   (RecordSet As RecordSet)
   RecordSet("LastName")  = MyLastName
   RecordSet("FirstName") = _
      MyFirstName
   RecordSet("ObjectID")  = ObjectID
End Sub
```

The object might not always know the value of the field "ObjectID," such as when a new object instance is being generated and ObjectID equates to a column that has the Counter attribute. Under these conditions, you can't determine the field's value until after the corresponding row has been inserted into the database. One way to determine the value of the ObjectID field is to instruct the data-aware Collection-object replacement to automatically set the new object's ObjectID property to the value of the Counter column.

RELATIONSHIPS IN A RELATIONAL DATABASE

The relational database doesn't inherently support object containment, but you'll need to work around that in order to continue the mapping process. Nearly every object model features an object containment hierarchy in which certain objects "contain" others. But the rules of table relationships and those of object containment are very different. You must resolve this in order to map the object containment hierarchy over the rela-

VB4

```
Private Sub Form_Load()

  Dim SampleEmployee As New Employee
  Dim SampleCustomer As New Customer
  Dim SampleProduct As New Product

  Set Database = _
     OpenDatabase(App.Path & "\MyDB.MDB")

' retrieve the appropriate rows from the
' Employee Table and instantiate
' Employee objects
  SmallBusiness.Employees._
     InstantiateFromDatabase _
     Parent:=SmallBusiness, _
     SampleObject:=SampleEmployee, Database:=Database

' set the datEmployees.RecordSet to the
' RecordSet created within the
' DataAwareCollection
  Set datEmployees.RecordSet = _
     SmallBusiness.Employees.RecordSet

  SmallBusiness.Customers._
     InstantiateFromDatabase _
     Parent:=SmallBusiness, _
     SampleObject:=SampleCustomer, Database:=Database
  Set datCustomers.RecordSet = _
     SmallBusiness.Customers.RecordSet

  SmallBusiness.Products._
     InstantiateFromDatabase _
     Parent:=SmallBusiness, _
     SampleObject:=SampleProduct, _
     Database:=Database
  Set datProducts.RecordSet = _
     SmallBusiness.Products.RecordSet

End Sub
```

LISTING 1 ***Open for Business.*** *This example uses the data awareness of the Collection-object replacement to instantiate its contained objects from the database. At the conclusion of this event, the three data controls—"datEmployees," "datCustomers," and "datProducts"—are ready for use.*

tional database model.

To illustrate this, let me explain these opposing concepts. In the relational world, "children" know about their "parents" (because foreign keys are located in the "child" rows), and the parent rows do not know who their children are. In the object world, the reverse is true: container objects (the parent objects) know about their contained objects (the child objects), and the contained objects do not know who their containing objects are.

In VB4, the Collection object is the primary mechanism for implementing object containment. To retain the Collection object's ability to implement the object containment hierarchy while adding relational database mapping features, I decided to store all object containment details in a separate, dedicated data store, independent of both the parent and child data stores, similar to the "many-to-many" resolution tables used in relational database design. That is, for each instance of object containment in the BOM, the dedicated data store records the containing object's type and ObjectID along with the contained object's type and ObjectID.

The solution automatically implements the object containment hierarchy by encapsulating the process of generating and executing the SQL statements needed to automatically retrieve the associated rows from the appropriate data source of the contained objects. It coordinates with the class module to instantiate objects of the appropriate class and populate those objects with data from the returned rows. It then gathers the contained objects and returns them to the application as a Collection object.

As it would apply to the SmallBusiness example in Chapter 7 of the *VB Programmer's Guide*, instances of my Collection-object replacement or similar solution would exist in the SmallBusiness class module as container objects for the collections of the SmallBusiness's Employees, Customers, and Products. Of course, with these new capabilities you could easily expand the example to store its data in a database with tables corresponding to the contained class modules (see Figure 1).

As the instances of the data-aware Collection-object replacement are referenced for the first time, they perform the necessary SQL-related functions to retrieve the data rows, instantiate and populate objects, and return the collection of contained objects to the application.

In this example, the Form_Load event procedure automatically instantiates the Employees, Customers, and Products for the SmallBusiness (see Listing 1). At the conclusion of this event, all of the necessary objects have been instantiated and the three data controls—“datEmployees,” “datCustomers,” and “datProducts”—are ready for use.

At this point, you should have a high level of integration between the VB4 class module and the VB4 database through successful mapping of objects to the relational database. Now you need to integrate the VB4 class module with the VB GUI.

GIVE THE GUI WHAT IT NEEDS

You could reason that if you can map a class module to a row in a record set, then you can map the data-aware Collection-object replacement to the entire record set. Managing RecordSet as a Collection object would help you use object-oriented techniques to manage VB controls that rely on RecordSet or that are bound to VB Data controls.

By using the data-aware replacement for the Collection object, you can design and develop the BOM without thinking about the needs of the database or GUI. Then, as you add the GUI and database components, you can exploit the capabilities of the data-aware Collection object to satisfy the needs of the VB RecordSet, Data control, and any bound controls.

The data-aware Collection-object replacement, called the “DataAwareCollection” here, should include methods for supporting the RecordSet property of the Data control in an object-oriented manner. An example of this code is:

```
Dim MyCollection As
DataAwareCollection
    .
    .
    .
Set MyDataControl.RecordSet = _
    MyCollection.RecordSet
```

Now you have an excellent opportunity to provide various other object-oriented RecordSet manipulation methods such as

"Add," "Remove," "Replace," "Item," "Refresh," "InstantiateFromDatabase," "InitializeFromRecordSet," "InitializeRecordSet," and "(Set) RecordSet."

At this point, you've provided a high level of integration between the class module and the GUI through the data-aware Collection-object replacement and its compatibility with the class module and the data control.

Through an interface of public methods, the data-aware Collection-object replacement exposes its underlying RecordSet object so an application can use it. As shown in the code example, an application could use the exposed RecordSet object to set the RecordSet property of one of its data controls. This would essentially allow the application to have a bound control (say, a DBGrid) bound to that data control, and that bound control would be the GUI equivalent of the Collection.

In effect, the BOM is thinking "a collection of employees of the company" (note the collection implication) while the GUI is thinking "a DBGrid showing the employees of the company." It's reasonable to assume that any VB controls bound to such a data control should be able to receive similar benefits from this object-oriented approach, typically in the form of object-oriented wrappers designed especially for those VB controls.

THE RELATIONAL DATABASE DOESN'T INHERENTLY SUPPORT OBJECT CONTAINMENT.

For example, you could construct a list box wrapper to serve as a special interface between your data-aware Collection-object replacement and a given VB GUI control type. In this case, the wrapper would provide an interface that lets you use the contents of your data-aware Collection-object replacement to populate the associated list box. So you could have a data-aware collection of employees, a list box intended to display employees, and a wrapper that interfaces the list box with the contents of the data-aware collection.

Methods would exist to populate the list box with data from the contained objects, to get or set the objects corresponding to the selected items in the list box, and so on. This process provides an object-oriented alternative for the conventional, GUI-centric means of managing a ListBox where the application must treat the ListBox as a holder of strings rather than objects. Similarly, you could build wrappers for the ComboBox, DBGrid, and DBList that support an object-oriented method of managing those VB Controls, as well.

The code discussed in this column is available online in a file called PC0496.ZIP. Download the file from *VBPJ's* Development Exchange on the World Wide Web at http://www.windx.com, or from the *VBPJ* CompuServe Forum, or MSN site. For details, see "How to Reach Us" in *VBPJ's* Letters to the Editor.