Microsoft Access Forms—All Class

Garry Robinson

If object-oriented development seems foreign to you, it shouldn't. All forms are defined in class modules, and all executing forms are objects. Garry Robinson shows how to take advantage of this to create classy forms.

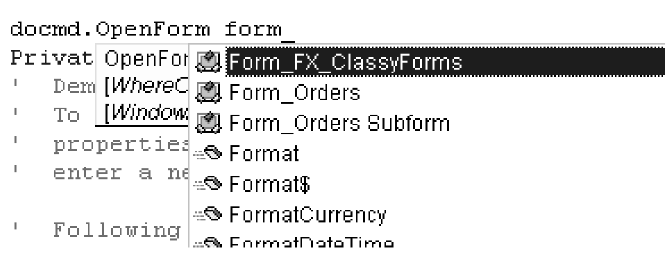
WHEN Access 97 came out, one of its hidden features was that form objects were now actually class modules. I was under-impressed. Five years later, and guess what—it finally dawned on me that I was using Access forms more and more as class modules. This has gradually come about as my clients started requiring more intuitive interfaces. This article outlines ways that you can take advantage of class module features that I've used over the past few years. You'll note as you read the article that I haven't gone really overboard on technicalities but instead have concentrated on showing you ways to get even more mileage out your most valuable forms.

My most-used trick

Whenever I add a control to a form that will open another form in Access, I avoid using the Button Wizard as all the error code that it generates irritates me. Instead, I make a blank button by canceling the wizard. Then, as usual, I find that I don't quite remember the name of the form that I want to open. So I'll type the following code and hit the space bar to show a list of forms (see **Figure 1**):

```
Docmd.openForm  form_
```

This list is actually all the forms in that database that have code behind the form. These are all stored in your database as class modules. And guess what—you can use these modules as objects.



```
docmd.OpenForm form_
Privat OpenFor  Form_FX_ClassyForms
'  Dem [WhereC  Form_Orders
'  To  [Window  Form_Orders Subform
'  properties   Format
'  enter a ne   Format$
'  Following    FormatCurrency
                FormatDateTime
```

Figure 1. Retrieve all the form class objects in your database.

Back to my trick. From the list of forms, I can select the form that I want. I then delete the "form_" before the form name, and I now have my form name. If the form that you want isn't in this list, you must ensure that the HasModule property of the form is set to True. This can make for slower loading of the form if it has no code behind it, so beware about getting carried away with setting this property to True.

The form as a class module

When you open a form with the DoCmd.openForm method, there's a limited number of things that you can do to the form:

- Show the form in datasheet or normal mode.
- Filter records behind the form.
- Display the form in Dialog mode.

If you're really tricky, you can use the OpenArg argument to pass all sorts of clever text strings to the form. These clever text strings can then be manipulated in the Form_Open event to manage the form.

There's a simpler alternative to these limited choices and complicated code: Use the form as an object. You can then manipulate the form with code like this:

```
DoCmd.OpenForm "Orders"
With Form_Orders
  .AllowEdits = True
  .RecordSelectors = True
  .NavigationButtons = False
  .Caption = "Smart Access Demonstration"
  .CustomerID.StatusBarText = "My Message"
End With
```

As you can see, many of the things that you might set at design time (for example, turning the record selectors and navigation controls on and off) can be handled easily in code. This is very useful because it allows you to modify the way the form looks while it's being opened rather than at design time. For instance, navigation buttons are useful for managing many records and a helpful addition to the user interface. But if all you want the form to do is allow the user to add one new record, then you can turn the navigation buttons off as you open the form and reduce the clutter in your user interface.

There's nothing more to the manipulation of the properties for your form than adding a code block after your OpenForm as I've done for the Orders form in my previous example:

```
With Form_Orders

End With
```

Now you can type a period inside the With block, and the following programmable items for the form will appear:

- All the form's methods (Undo, SetFocus, Requery, Refresh, and Repaint)
- An abundance of properties, including Dirty, Hwnd, Cycle, and DatasheetFontName
- All the text boxes, combo boxes, and other controls that exist in the form

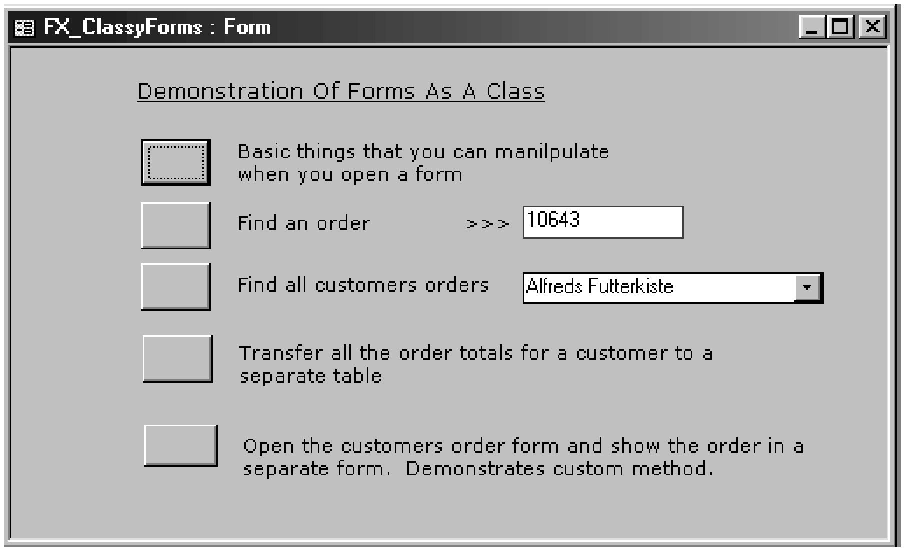
If you select a control for the form, you can manipulate the properties of that control. Available control properties include its default value, font characteristics, and control tip—plus all the other properties that you're used to setting at design time.

Data entry modes

The primary reason I've been manipulating the properties of forms in code is to set up different styles of interfaces for my end users without generating multiple copies of what's basically the same form. The following example demonstrates how the user interface can be switched to either Add or Edit mode according to a field on another form (shown in **Figure 2**). If the user leaves the order number field blank, the next form is opened in DataEntry mode. If the order number isn't blank, then the form receives the focus in code, and the FindRecord method of the DoCmd object is used to find the specified order.

The code for this is shown in the next snippet. While there's not much difference between the two versions of the form, the differences reflect things that make the user interface less confusing than it could be and tailor the second form for its purpose. For example, the record navigation control is on when an order exists and off when you're adding a new order. Further on, I'll show you how the form's recordset can be used to determine whether an order actually exists.

```
Dim UserResponse As Variant
```



FX_ClassyForms : Form

Demonstration Of Forms As A Class

Basic things that you can manilpulate when you open a form

Find an order >>> 10643

Find all customers orders Alfreds Futterkiste

Transfer all the order totals for a customer to a separate table

Open the customers order form and show the order in a separate form. Demonstrates custom method.

Figure 2. The demonstration form.

```
DoCmd.OpenForm "Orders", , , , , acHidden
With Form_Orders
  If IsNull(Me!orderReq) Then
      .AllowAdditions = True
      .DataEntry = True
      .NavigationButtons = False
      .Visible = True
  Else
      .Visible = True
      .DataEntry = False
      .NavigationButtons = True
      .OrderID.SetFocus
      DoCmd.FindRecord Me!orderReq, acEntire, _
                , acSearchAll
  End If
End With
```

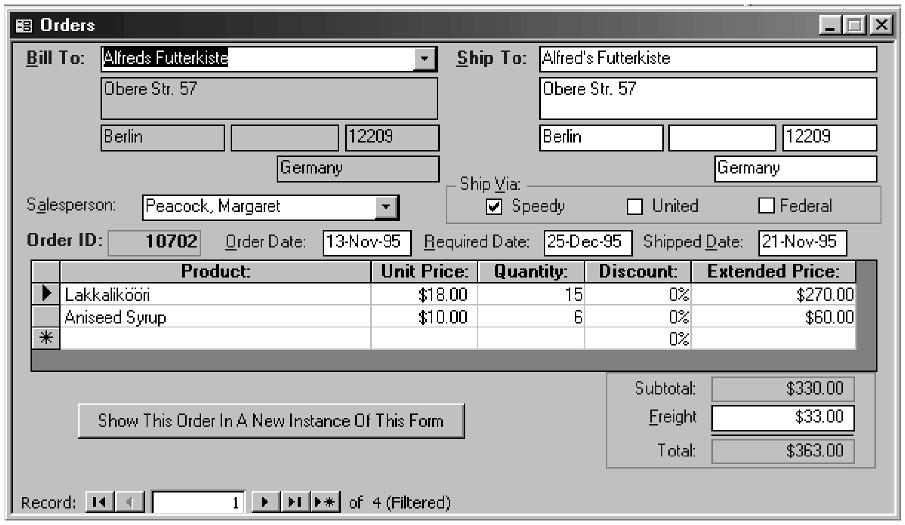
Your own filters

The DoCmd object's OpenForm method allows you to add your own Where clause or query to filter your data. However, it's often cleaner and more flexible to program filters using the filter properties of the form. This code shows how easy it is to modify the Filter property of another form to have it show only the records for a particular customer:

```
DoCmd.OpenForm "Orders", , , , acFormAdd, acHidden
With Form_Orders
 .Filter = "CustomerId = '" & Me!CustomerReq & "'"
 .FilterOn = True
 .NavigationButtons = True
 .Visible = True
End With
```

A word of caution, if performance is a big issue: When you use DoCmd.openForm and don't apply either a filter query or a Where clause, the form will open with a recordset that retrieves all the data in the recordsource. Applying the filter in the OpenForm method ensures that the recordset behind the form is filtered before it's displayed. If you're using the FilterOn and Filter properties of the form as I do here, it's wise to open the form using the acFormAdd constant in the

OpenForm's dataMode argument (as I've done in my example). This opens the form with no data being displayed. This is fast, and your data will be displayed when you turn the filter on. I also like to hide the form from the user while the form is being manipulated by using the OpenForm's acHidden property. The form is made visible by setting the form class's Visible property at the end of my manipulation.



Orders

Bill To: Alfreds Futterkiste
Obere Str. 57
Berlin | | 12209
Germany

Ship To: Alfred's Futterkiste
Obere Str. 57
Berlin | | 12209
Germany

Salesperson: Peacock, Margaret

Ship Via: [✓ Speedy] United Federal

Order ID: 10702 Order Date: 13-Nov-95 Required Date: 25-Dec-95 Shipped Date: 21-Nov-95

Product:	Unit Price:	Quantity:	Discount:	Extended Price:
Lakkalikööri	$18.00	15	0%	$270.00
Aniseed Syrup	$10.00	6	0%	$60.00
			0%	

Subtotal: $330.00
Freight: $33.00
Total: $363.00

Record: 1 of 4 (Filtered)

Figure 3. The slightly modified Northwind orders form.

Look, Mum—no hands

If ever I had a golden rule for computing, it would be that you never want to code the same thing twice, especially if you're being paid for it. I ran into this issue with a programming buddy where we realized (after spending hours trying to calculate a total on a form using sub forms and other bits of code) that we then had to rewrite all that code to use those results in a report. In this case, I realized that the form actually contained all the business logic for the calculations.

This brings me to the next example where I show you how to transfer calculations from the good old Northwind Orders form to a temporary table (see Figure 3). The fields that I'll transfer are the OrderId, subtotal, freight, and the total calculated on the form.

I start by opening the Orders form and filtering the customer records. In the sample code, I call the code that's behind the Find Customers button. Never forget that code under a button can be reused elsewhere since it is, after all, just a subroutine. After the filter, I have an open form with only the orders for the one customer I was filtering for.

I then manipulate those records by using the form's Recordset property. This is really exciting, because you can walk through records using all the familiar MoveNext and MoveFirst methods of a recordset. You can also test to see whether the recordset has no records and stop any action. In the following code, I also test the RecordCount property when opening the form to see whether any data exists:

```
With Form_Orders
  .Recordset.MoveFirst
  DoCmd.SetWarnings False
  If .Recordset.RecordCount = 0 Then
    MsgBox "No orders were found for " & _
    Me!CustomerReq, vbOKCancel, "Try Again"
    DoCmd.Close acForm, "Orders"
    GoTo exit_cmdTotals_Click
  Else
```

Unfortunately, this doesn't work in Access 97 because the form class module doesn't have a Recordset property.

Now that I've established that I have some records, I loop through the records for that customer. You may want to do this with the form hidden once you move the application to production. As I'm about to add the order totals to a blank table, I need to clear the table first and turn off all the Access insert warnings:

```
DoCmd.SetWarnings False
DoCmd.RunSQL "Delete from MyOrders"
Form_Orders.Visible = True
```

Now it's a matter of looping through the form's recordset, which has the effect of showing each record on the form (a good reason to make the form invisible unless you think that your users will enjoy the show). It seems to be necessary to use the form's recalculation method for each record to ensure that calculated fields are populated.

In this example, I've used a SQL Insert statement to add the data to my temporary table. All of this code occurs in the form that opened and is controlling the Orders form and not in the Orders form itself. As you can see in the next code block, I've referred to the Orders form's subtotal field by treating it as a property of the Orders form. The full property that I reference is Form_Orders.Subtotal.Value. To move to the next record for this customer, I simply use the MoveNext method of the recordset. The form then shows the next record:

```
 Form_Orders..Recalc
 sqlStr = "INSERT INTO MyOrders " & _
   "( OrderID, SubTotal, Freight, Total ) values " & _
   "(" & Form_Orders.OrderID & "," & _
   Form_Orders.Subtotal & "," & _
   Form_Orders.Freight & "," & _
   Form_Orders.Total & ")"
  DoCmd.RunSQL sqlStr
  Form_Orders.Recordset.MoveNext
Wend

DoCmd.OpenTable "MyOrders"
```

Open the same form twice

On the Orders form (see Figure 4), I've put a button to show the form again. You can actually display a copy of the current form without copying it to a new name in the database container. You can do this because the form is a class and can be instantiated as a new object. The following code shows how I can make a copy of the form, filter it for the current order, and modify a few properties so that the form looks different from the current version. I find this technique is useful for comparing two complex records using a standard form view:

```
Static lastTop As Long, lastLeft As Long
Set frmOrders = New Form_Orders

With frmOrders
  .Visible = True
  .Filter = "orderId = " & orderReq
  .FilterOn = True
  .Caption = "Filter:  " & .Filter
  .Detail.BackColor = vbWhite
  lastTop = lastTop + 100
  lastLeft = lastLeft + 100
  DoCmd.MoveSize lastTop, lastLeft
  .cmdCopyOrder.Visible = False
End With
```

Of course, there are a few tricky things to realize about this new form object. You'll have to explicitly set the Visible property of the new copy of the form to True to get it to display. Even more perplexing is that, in your first attempt, you'll probably write the code so the variable used to refer to the form immediately drops out of scope once the form has been displayed. Once the variable is out of scope, the copy of the form is destroyed. This is what happens if you use a local object variable to refer to the form like this:

```
Sub CopyOrder
Dim frmOrders As Form_Orders
  Set frmOrders = New Form_Orders
  With frmOrders
    .visible = true
  End With
End sub
```

The solution is to declare the variable that refers to the form object so that it remains in scope even when the subroutine has completed running. You can achieve this by declaring the variable as at the module level of the form:

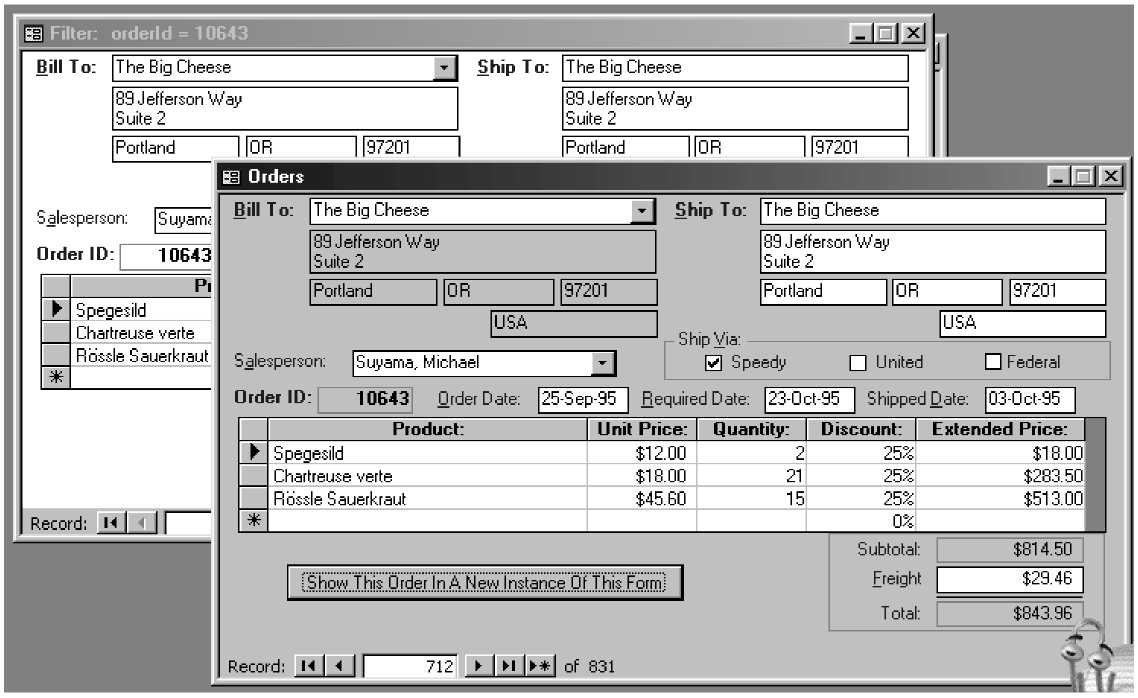
```
Dim frmOrders As Form_Orders

Sub CopyOrder
  Set frmOrders = New Form_Orders
  With frmOrders
    .visible = true
  End With
End sub
```

Another solution is to declare the variable as static:

```
Static frmOrders As Form_Orders
```

Declaring the variable as static has an interesting effect: If you then run the same code again, the form seems to save the record that you're editing and then refresh the same object with the new properties that you've set for the form. This means that if you want to maintain multiple versions of the class objects, you'll need to keep an array of form objects. An even more sophisticated approach is to make your own collection of form objects and manage the forms in that special collection:


Filter: orderId = 10643

Bill To: The Big Cheese
89 Jefferson Way
Suite 2
Portland OR 97201

Ship To: The Big Cheese
89 Jefferson Way
Suite 2
Portland OR 97201

Salesperson: Suyama
Order ID: 10643

Spegesild
Chartreuse verte
Rössle Sauerkraut

Record:

Orders

Bill To: The Big Cheese
89 Jefferson Way
Suite 2
Portland OR 97201
USA

Ship To: The Big Cheese
89 Jefferson Way
Suite 2
Portland OR 97201
USA

Ship Via: [✓ Speedy] United Federal

Salesperson: Suyama, Michael

Order ID: 10643 Order Date: 25-Sep-95 Required Date: 23-Oct-95 Shipped Date: 03-Oct-95

Product:	Unit Price:	Quantity:	Discount:	Extended Price:
Spegesild	$12.00	2	25%	$18.00
Chartreuse verte	$18.00	21	25%	$283.50
Rössle Sauerkraut	$45.60	15	25%	$513.00
			0%	

Subtotal: $814.50
Freight: $29.46
Total: $843.96

Record: 712 of 831

Figure 4. Create a new instance of the Orders form and display the current order.

```
Dim colOrders As Collection

Sub CopyOrder
Dim frmOrders As Form_Orders
  Set frmOrders = New Form_Orders
  With frmOrders
    .visible = true
  End With
  colOrders.Add Form_Orders
End sub
```

So when does this second form actually close down or go out of scope? The form will close down if the user closes it down manually, and it will close down if the form that holds the static or module-level variable is closed down. To have the form stay alive for the life of your application, declare the variable that refers to the form as Public in a module:

```
Public frmOrders As Form_Orders
```

Your own methods

Everything has been pretty exciting thus far, but there's more. You can make your forms really clever by exposing your private subroutines and functions as Public. For example, when I made a new instance of the Orders form, I decided that it would be cool to not only open the Orders form but to use a special method of that form to display that order in yet another form as well.

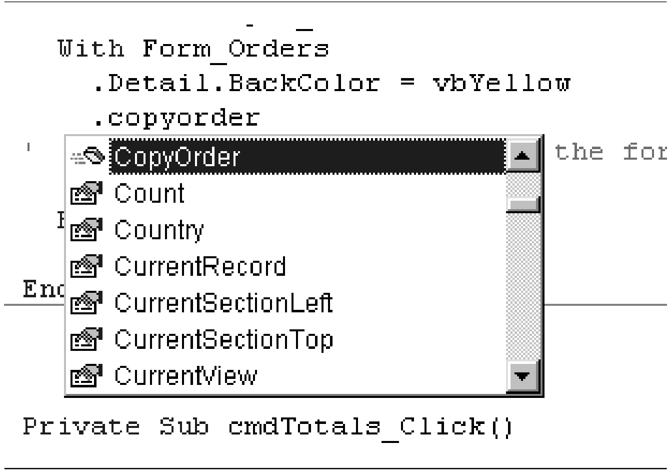
I managed this by making the CopyOrder routine a public subroutine. I also added an optional argument to this subroutine so that I could pass an order number of my choosing into the subroutine:

```
Public Sub CopyOrder(Optional ShowOrderID As Variant)

Static lastTop As Long, lastLeft As Long
Static frmOrders As Form_Orders
Dim orderReq As Variant

If IsMissing(ShowOrderID) Then
   orderReq = Me!OrderID
Else
  orderReq = ShowOrderID
End If

End sub
```



```
   With Form_Orders
     .Detail.BackColor = vbYellow
     .copyorder
 '                                    the for
  I
 End
  Private Sub cmdTotals_Click()
```

Figure 5. Once you set up a form class method, it immediately becomes visible in IntelliSense.

As you can see in **Figure 5**, this public function now appears as a new method of the Form_Orders form. I now can call this new method using an order number like this:

```
With Form_Orders
  .CopyOrder Me!orderReq
End With
```

It's that easy to turn a subroutine into a method. Any public function also acts as a method, except that they can return values after they've completed.

You can also add public properties to your form. The easiest way is to choose Insert | Procedures from the menu while working in a code module behind a form. You can also write the code yourself. To create a property that can be both read and written to, you must write a Property Let (which will be run when someone tries to set your property) and a Property Get (which will be run when someone tries to read your property). These property routines allow the user to change the OrderNum variable inside the form through a property called OrderNumber:

```
Dim OrderNum As String

Public Property Let OrderNumber(Order_Numb As String)
    OrderNum = Order_Number
End Property

Public Property Get OrderNumber() As String
  OrderNumber = Order_Num
End Property
```

As you can see, a Property Let looks very much like a subroutine that accepts a single parameter; a Property Get looks like a function that returns a value.

Code that uses the property might look like this:

```
Dim frmOrders As Form_Orders

Sub CopyOrder
  Set frmOrders = New Form_Orders
  frmOrders.OrderNumber = Me!OrderID
```

Finally, you can have your form fire events back to the code that created it. This code, placed inside the Form_Orders form, defines an event called OrderNotFound:

```
Event OrderNotFound()
```

To signal that the order isn't found, the Form_Orders form would use the RaiseEvent command with the name of the event:

```
RaiseEvent OrderNotFound
```

In the code that creates the Form_Orders form, you have to declare the variable that refers to the form with the WithEvents keyword in order to catch the events.

Continues on page 23

XML Web Reports...

Continued from page 8

Although I didn't run any time trials, it's reasonable to conclude that it's faster to create a single XML file from a query than to create both XML and XSL files from a report.

I have to admit that, though I was excited to hear that Access 2002 would include support for XML, I was at first hard pressed to find a real-life application for it. Once I began to create XML reports for Web apps, it became clear to me that Access 2002 could greatly simplify the process. Now, even if I use a different method to refresh the XML files used in the transform, I'll continue to use the ExportXML method to generate the complex XSL files required to produce rich Web reports. ▲

DOWNLOAD **XMLREPORT.ZIP at www.smartaccessnewsletter.com**

Danny J. Lesandrini, a Microsoft Certified Professional in Access, Visual Basic, and SQL Server, has been programming with Microsoft Access since 1995. He maintains a Web site containing Access-related code solutions at http://datafast.cjb.net and replies to all questions and comments sent to him via email. datafast@attbi.com.

Microsoft Access Forms...

Continued from page 13

Once you do that, you can respond to the events by writing an event procedure whose name consists of the variable name and the event name:

```
Dim WithEvent frmOrders As Form_Orders

Sub CopyOrder
  Set frmOrders = New Form_Orders
  frmOrders.OrderNumber = Me!OrderID
End Sub

Sub frmOrder_OrderNotFound()
..code to handle order number not found
End Sub
```

The download database

The Download file for this article (available at www.smartaccessnewlsetter.com) consists of a database in Access 2000 format. I originally started doing all the samples in Access 97 but stopped because Access 97 wouldn't support recordsets behind the form. The Access 97 version of this code is actually the Access 2000 version saved back to Access 97. Most of the sample code works, but the last two options on the demo form (which depend on the Recordset property of the form) will fail under Access 97.

Your Access forms are pretty smart objects and are one of the reasons why Microsoft Access is such a configurable tool. Now that you've seen how the form can be manipulated as an object, you can make smart forms to suit your clients' user interface requirements. The "form as class object" also includes all the form properties and events that you're used to manipulating. Now you can manage these properties in code before you expose the form to your user. In some cases (for instance, recordset manipulation), you can even leverage the considerable work that you've put into your forms to transfer information to tables. You can then use those forms in reporting and other activities. Forms are a class act, so why not start using them as the class objects that they truly are? ▲

DOWNLOAD **FORMCLSS.ZIP at www.smartaccessnewsletter.com**

Garry Robinson works for GR-FX Pty Limited, a company based in Sydney, Australia. If you want to keep up-to-date with the his latest postings on Access issues, visit his company's Web site at www.gr-fx.com or sign up for his Access e-mail newsletter by sending a blank e-mail to tips@gr-fx.com. He's recently made available a library of code and forms that he reuses in all his projects. Read about it at www.vb123.com/toolbox. garry@gr-fx.com.

Useful Further Reading and Resources

- Search your Access Help for "Program with Class Modules." Choose the topic "Create a class module that's not associated with a form or report." The links for this page take you to information about programming form class objects.
- *The Access 2000 Developers Handbook* (Desktop Edition) covers this topic in great detail. Read my review at www.vb123.com/books.