Display Text with Property Settings

by Karl E. Peterson

Click & Retrieve Source CODE!

Q SEE DESCRIPTIONS WITH VALUES

During my first effort to design an ActiveX control using VB 5.0, I ran into a small problem. I have an integer-type property on an ActiveX control project that can be edited through a property page. On the property page, I fill a combo box with the acceptable set of property values:

```
cboProperty.AddItem "0 - Descriptive text for 0"
cboProperty.AddItem "1 - Descriptive text for 1"
cboProperty.AddItem "2 - Descriptive text for 2"
```

When the value is changed through combo box selection, I use the ListIndex of the combo box to apply the new value to its corresponding property in my control. The problem is that when I return to the main property sheet to inspect the value currently set for the property, I see only the Integer value without its corresponding text. Ideally, I would like to see both. For example, VB's Alignment property displays something like this:

```
0 - Left align
1 - Right align
2 - Center
```

How can I accomplish this?
—Robert Mayer, Memphis, Tennessee

A

This situation has tripped up a number of folks. The answer isn't intuitive, but it is extremely simple to implement. You didn't show your exact code, so I'll use the Alignment example you brought up. To replicate that, you first need to expose the constants through a Public Enum:

```
Public Enum AlignmentTypes
   [Left Align] = 0
   [Right Align] = 1
   Center = 2
End Enum
```

The first two constant names are enclosed in brackets. Using this style allows spaces to appear in the constant names. The spaces make the property window appearance more natural, but introduce a problem for users who would like to assign these constant values directly within their code. As in the Enum, they have to enclose the constant name within brackets in their code:

```
UserControl1.Alignment = [Left Align]
```

You can ease this pain by declaring the Property as the enumerated type. As your users code for this property, VB5 presents them with a drop-down list of the potential values they're allowed to assign. Choosing one of the listed values automatically encloses it in brackets if it contains a space. There is just one gotcha—VB won't validate incoming values to ensure they're within the range of the enumerated constants. You must test and act accordingly within the Property procedure:

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the Visual Basic Programmer's Journal *Technical Review and Editorial Advisory boards. Based in Vancouver, Washington, he's also an independent programming consultant specializing in ActiveX controls. Karl coauthored* Visual Basic 4 How-To *from Waite Group Press and contributes to various journals. Online, he's a Microsoft Developer MVP and a section leader for both* VBPJ *online forums. Contact Karl at karl@rtc.wa.gov.*

This is your forum for addressing the intricacies of Visual Basic. Send your questions, clever tips, and techniques. Visual Basic Programmer's Journal *will pay $25 for any submission, tip, or question we print. If your submission includes code, please send it electronically. Please include your mailing address with your submission. Mail submissions to Q&A Columnists, c/o Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA, USA, 94301-2500. CompuServe: 74774,305. Internet: vbpjedit@fawcette.com.*

```
Public Property Let Alignment_
   (ByVal NewVal As AlignmentTypes)
   If NewVal = Center Or _
      NewVal = [Left Align] Or _
      NewVal = [Right Align] Then
         ' store new value and
         ' alter behavior to match
   Else
         ' illegal value
   End If
End Property
```

The sharp-eyed amongst you are saying by now, "I could've saved a lot of trouble by using one of VB's standard enumerated types." That's true. And in general, it's an excellent method to save a lot of keystrokes. In this case, I could have used AlignmentConstants instead of AlignmentTypes as the Property type and VB would have automatically offered vbCenter, vbLeftJustify, and vbRightJustify as the available options. The Enum would not have been needed. You can see how simple that would make offering a property such as MousePointer, which has more than a dozen standard choices. Of course, this was only an example.

Q ACCESS FIELD DESCRIPTIONS

Microsoft Access 2.0 and 7.0 have a "comments" or "descriptions" section for each field in a database. When programming in Access, this field shows up on forms. How can I access this in VB code? I would like to use the comments for tooltip help, or simply to provide the users of my database apps with more information on the field they are entering data into. Is there any way to do this?
—Kevin Pisarsky, Akron, Ohio

A

Like you, I was initially stumped by this question. A quick browse of the Microsoft Knowledge Base proved discouraging. Article Q109136, "Basic Cannot Get Description Shown in Access Table Design View," seemed to say this was impossible. However, closer examination showed this article was written for VB3 and Access 1.x. I did a little snooping around to see what I could uncover. My thanks to Joe Maki (a Microsoft Access MVP) for reminding me, once again, not to believe everything I read. Joe offered this snippet to extract all properties for any given field:

```
Private Sub Form1_Click()
   Dim db As Database
   Dim td As TableDef
   Dim fld As Field
   Dim prp As Property

   On Error Resume Next

   Set db = OpenDatabase_
      ("F:\Prog\VB5\Biblio.mdb")
   Set td = db.TableDefs("Authors")
   Set fld = td.Fields("Author")
   For Each prp In fld.Properties
      Debug.Print prp.Name & ": " & _
         prp.Value
   Next
End Sub
```

I used the sample Biblio.mdb database, which ships with VB, for the test. To extract only the Description field, you can assign it directly:

```
Text1.ToolTipText = _
   td.Fields("Author").Properties_
   ("Description")
```

Alternatively, this syntax might be marginally faster, and is another alternative if it more closely matches your coding style:

```
Text1.ToolTipText = _
   td.Fields!Author.Properties!_
   Description
```

This tip comes with several caveats should you decide to use it. Foremost is that DAO raises an error if the property you seek doesn't exist. Appropriate error-checking is advised. You can find a complete example in the VB5 help file under the CreateProperty topic. A less complete, but still helpful, example is also included in the VB4 help file. This capability was added after Access 2.0 and the compatibility layer update, making it unavailable in VB3.

VB4 32-bit VB5

```
Option Explicit

Private wrd As Word.Application

Private Sub Command1_Click()
   With CommonDialog1
      .ShowOpen
      Call WordOpen(.Filename)
   End With
End Sub

Private Sub Form_Load()
   Call WordStart
End Sub

Private Sub Form_Unload(Cancel As Integer)
   If Not wrd Is Nothing Then
      wrd.Quit
      Set wrd = Nothing
   End If
End Sub
```

```
Private Function WordStart() As Boolean
   On Error Resume Next
   Set wrd = CreateObject("Word.Application")
   If Err Then
      MsgBox "Cannot Start Microsoft Word"
      Debug.Print Hex(Err.Number), Err.Description
   Else
      wrd.Visible = True
      WordStart = True
   End If
End Function

Private Sub WordOpen(ByVal Filename As String)
   On Error Resume Next
   wrd.Documents.Open Filename
   If Err.Number = &H800706BA Then 'Automation error
      Call WordStart
      wrd.Documents.Open Filename
   End If
End Sub
```

LISTING 1 ***Automating Microsoft Word.*** *This little sample illustrates how to open multiple documents in Microsoft Word without creating new instances for each document. This listing is the complete code from a form with just two controls: a common dialog and a command button.*

INTERMEDIATE Q&A

Q USE OLE AUTOMATION FOR WORD DOCUMENTS

I'm trying to run an application such as Word, using Shell from VB, to open a document file. When I finish viewing it, instead of killing it, I'd like to keep it around and use it to open a different document when I need it. My command looks like this:

```
MyAppID = Shell_
   ("C:\MSOFFICE\WINWORD\" & _
   "WINWORD.EXE " & strDocFile, _
   vbNormalFocus)
```

The problem is that it opens a copy of Word each time. After viewing five documents, I have five Word copies running.
—P. Bruce Sung, received by e-mail

A Your scenario presents two viable approaches, the simpler of which is to use the ShellExecute API rather than VB's Shell function. ShellExecute uses an existing instance of Word to open additional documents rather than starting a new instance for each document. For more details, see Knowledge Base article Q170918, "How To Use ShellExecute to Launch Associated File (32-bit)." Here's a quick example, using a common dialog control to pick the file:

```
Private Declare Function ShellExecute _
   Lib "shell32.dll" Alias _
   "ShellExecuteA" _
   (ByVal hwnd As Long, _
   ByVal lpOperation As String, _
   ByVal lpFile As String, _
   ByVal lpParameters As String, _
   ByVal lpDirectory As String, _
   ByVal nShowCmd As Long) As Long

Private Sub Command1_Click()
   With CommonDialog1
      .ShowOpen
      Call ShellExecute(Me.hWnd, _
         "open", .Filename, "", "", _
         vbNormalFocus)
   End With
End Sub
```

The other option, OLE Automation, is considerably more involved, but offers much more opportunity to manipulate the document(s) from within your app. If you want to do more than simply open a document and view it, this is the route worth investigating. I've worked up a simple sample applet that demonstrates the basic principles. To re-create it, open a new project, select "Microsoft Word 8.0 Object Library" from the Project-References dialog, and enter the code I've included (see Listing 1).

I chose to use early binding by declaring a variable explicitly as Word.Application rather than as Object, which serves to improve performance because VB can resolve external references before compiling. I declared this variable at the form level, so it would be readily accessible throughout the demo. In a real application, depending on your needs, you might use global scope for such a variable.

The demo starts by creating a new instance of Microsoft Word in the Form_Load event. The user can then press a command button to display a common file dialog. The program passes the selected file to the Word object in the WordOpen subroutine. Using the Open method of the Word object's Documents collection, a single line of code opens the file. Firmly believing in restoring a computer to the state I found it in, I coded the demo to shut down Word in the Form_Unload event.

It's been said before, but the advice is invaluable when you start experimenting with OLE Automation: get into the habit of pressing the F2 key to bring up the object browser. By adding the reference to the Word object library, you can view all exposed objects together with their associated methods, properties, and events. Although I rarely need this sort of operation and am not familiar with the Word object model, I threw together this demo in a short time with the help of the object browser.

Q ASKING FOR TROUBLE

I use VB5. I recently downloaded a VB3 decompiler from a VB Web site. I wanted to look at this really neat VB3 program's source code. Everything worked fine, but when I tried to view the FRM files in VB5, it said the files were in binary format and couldn't be viewed. Is there any VB3-to-VB5 converter of source code? If not, what can I do to use the code in VB5?
—Name withheld

A Wow. Where to start with this one? I have omitted the names you mentioned for your and their protection. I believe you are referring to a VB3 decompiler by Ash Rofail and Dr. Hans P. Diettrich (who uses the handle "DoDi" online).

I will answer your question because I believe there's a nugget of information in it that might interest others. Before doing that, however, I must say I find what you're doing to be totally uncool and probably illegal. I'm stunned you would publicly ask for help in stealing someone else's code. Perhaps it never occurred to you that this is what you're doing. Although it might seem like a great learning tool, consider how you'd feel if your competitors someday applied a similar tool to programs you had written.

Your question highlights the situation many were concerned about when this decompiler first became available. While there is *one* legitimate use of this product—recovering one's own source from an EXE—it is generally viewed as a method to facilitate theft. The authors, of course, have a different view, and have stated that the decompiler was written to highlight the *potential* for such a tool to be developed. Hmmm. In their defense, a method to protect applications from such tampering might be found by searching for "VBGuard" on the Internet (or, see "Master the Black Art of the VB Interpreter," *VBPJ* December 1996).

That said, the reason you can't load the recovered source is that VB3 used a binary format by default, and this tool does likewise with form files. Using the binary format has always been a dangerous choice, as one bad byte could corrupt an entire file. Microsoft dropped the binary format with the introduction of VB4. If you want to bring the old code forward, you have no choice but to load it into VB3, select Save File As from the File menu, and check the Save as Text box for each file in the project. You should also realize that if the form you're trying to convert uses any nonstandard VBXs, you can never load it successfully into VB5 without first obtaining updated OCX versions of those controls.

Code Online

You can find all the code published in this issue of* VBPJ *on The Development Exchange (DevX) at http://www.windx.com. All the listings and associated files essential to the articles are available for free to Registered members of DevX, in one ZIP file. This ZIP file is also posted in the Magazine Library of the* VBPJ *Forum on CompuServe. DevX Premier Club members ($20 for six months) can get each article's listings in a separate file, as well as additional code and utilities for selected articles, plus archives of all code ever published in* VBPJ *and* Microsoft Interactive Developer *magazines.

Display Text with Property Settings

Locator+ Codes

Listings ZIP file (free Registered Level): VBPJ1097

Listings for this article plus the fully built sample demonstrating OLE Automation with Word 97 (subscriber Premier Level): QA1097P