Who's My Parent?

by Karl E. Peterson

Q REFERRING TO PARENT PROPERTIES

Every now and then it would be convenient to design my objects so they could refer to the properties of the parent object that instantiated them. Given that there's no Extender-type object exposing this Parent property for class objects—as there is for UserControls—how can I pull this off?

A

That's a pretty tricky question, or rather a question that has only tricky answers. A couple different methods can accomplish your goal. Unfortunately, both have drawbacks.

The most straightforward method is simply coding a Parent property directly into your classes. After instantiating an instance of the class, set the Parent to Me—or whatever object should act as parent. At this point, the object purists out there must be howling. Yes, doing this breaks all the "rules" of encapsulation. Yes, you no longer have "pure" objects. But so what?

The fundamental problem with this scenario is that you've now set up a circular reference; the parent holds a reference to the child, and the child holds a reference to the parent. And in doing so, you've increased the reference count on the parent object. If the design of your parent object calls for disposing of its objects during its termination event, you find yourself in a Catch-22 because the Terminate event never fires as long as the parent reference exists in the child.

Cleaning up this mess requires that you de-couple this relationship before the parent's termination. You might set the child's Parent property to Nothing, thereby decrementing the parent object's reference count. Just thinking about what exactly would trigger this action is enough to drive an application designer crazy.

Another approach is to steal an uncounted reference to the parent object with a thoroughly slimy hack—the best kind, right? You can use RtlMoveMemory—commonly aliased as CopyMem or CopyMemory—to borrow an instance of your parent object as the need arises. The ideas behind this technique could consume a whole column (see William Storage's Programming with Class column, "Managing Dependent Objects," *VBPJ* February 1996). In short, code your Parent Let property to accept a Long rather than an instance of the parent object. When setting the Parent property, use ObjPtr(Me) to pass a pointer to the object, rather than just Me, which passes the actual object.

In the child object, store this pointer in a normal Long variable. When you need to call a method or set a property of the parent, use CopyMem to assign the stored object pointer to a locally declared variable of the same type as the parent, call the needed method, then clear the stolen pointer (see Listing 1). Now your scheme of having the parent destroy all its objects during termination should work without a hitch.

As a side note, the ObjPtr function isn't in the VB help files. For obvious reasons, documenting this function would greatly increase the burden on Microsoft's VB support team. Although this support burden is frequently used as a rationale for not providing powerful language constructs, in this case Microsoft slipped a gem into the product.

Q SHELLING A SHORTCUT

In Windows 95, when you click on a shortcut icon or an item under the Start menu, it appears that you are actually activating a Link file (LNK extension) that runs the program. How can I activate a LNK directly under VB? The Shell function appears to be unable to run LNK files because it doesn't recognize a LNK file as an executable, raising an error instead. Could an API call run the LNK files? I can see possible advantages to running the LNK file instead of using Shell to run the EXE.

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. In addition to contributing to various journals, Karl coauthored* Visual Basic 4 How-To, *from Waite Group Press. Online, he's a Microsoft MVP, and a section leader in both* VBPJ *online forums. Find more of Karl's VB samples at http://www.mvps.org/vb.*

Ask the VB Pro provides you with free advice on programming obstacles, techniques, and ideas. Read more answers from our crack VB pros on the Web at http://www.inquiry.com/thevbpro. You can submit your questions, tips, and ideas on the site, or access a comprehensive database of previously answered questions.

VB5

```
' *********************************
'  Example of code within Parent
' *********************************
Private Sub Form_Click()
  Dim cls As Class1
  Set cls = New Class1
  cls.Parent = ObjPtr(Me)
  cls.SomeMethod "Gotcha!"
  Set cls = Nothing
End Sub

' *********************************
'  Example of code within Child
' *********************************
Private Declare Sub CopyMem Lib "kernel32" Alias _
  "RtlMoveMemory" (Destination As Any, Source As Any, _
  ByVal Length As Long)

Private m_Parent As Object
Private m_lpParent As Long

Public Property Let Parent(ByVal NewVal As Long)
  m_lpParent = NewVal
End Property

Public Sub SomeMethod(ByVal SomeText As String)
  If m_lpParent Then
     ' Steal a reference to the parent object
     ' by copying a pointer to it into a local
     ' object variable.
     Call CopyMem(m_Parent, m_lpParent, 4)
     ' Set a meaningless property of the parent,
     ' just to show it works.
     m_Parent.Caption = SomeText
     ' Finally, clean up the stolen reference.
     Call CopyMem(m_Parent, 0&, 4)
  End If
End Sub
```

LISTING 1 ***Avoiding a Circular Reference.*** *Here's the most meager example possible of stealing a reference to a parent object from within a child object. A form creates an instance of the class, and passes a pointer to itself as the Parent. When your program calls the class's SomeMethod method, it copies this pointer into a local object variable, which your program can then use to call methods on the form. Neglecting to clear this stolen reference is fatal.*

A Sure; I can see times where this would be appropriate. You're correct that VB's Shell function won't do the job here. What you're looking for is an old standby—ShellExecute. This API is generally used to start the application associated with any given data file. For example, passing the URL to an HTML file causes your default browser to fire up and load that document.

In this case, you pass the path and file name of the shortcut file to ShellExecute (see Listing 2). Don't worry about most of the other parameters for this function, because they'll all be overridden by the setting stored in the shortcut.

Q **PREVENTING FOCUS**

I don't want a certain text box to appear disabled, but I also don't want users to be able to copy information from the control onto the clipboard. So setting the Enabled and/or Locked properties isn't what I'm looking for. Is there an API that prevents a text box from receiving focus without altering its appearance?

A First, to answer your question, no, there really isn't an API designed to do what you need. This column often dwells on API solutions, but sometimes getting into that mindset can keep you from using techniques that are readily available right within VB.

One method I've used when I needed this behavior is to place the text box within another container control—such as a frame or picture box—and set the container's Enabled property to False. This disables the text box, but with no change in its appearance. You can camouflage the container control by either setting its border to None or sizing it to exactly the same dimensions as the text box.

That said, if you *never* want to allow editing with this text box, why not simply use a label control instead? By setting the BackColor to vbWindowBackground and the BorderStyle to "1 – Fixed Single," a label control looks virtually identical to a text box (sans scrollbars, of course).

Q GETTING A WINDOW HANDLE

I would like to obtain the window handle for the taskbar clock. The Spy++ utility shows the class name for this window as "TrayClockWClass," but calling FindWindow on that class name always returns zero. Where am I going wrong?

A

I'm not sure I even *want* to know exactly why you're after this window handle, but your question provides the opportunity to highlight a useful, yet little-known, API function. It might come as a surprise to many that FindWindow returns the handles of only top-level windows, and is useless for finding child windows. The first time this tripped me up; I had to read the docs several times before whacking my own forehead when I figured it out.

The Spy++ utility—provided on the VB5 CD—shows that the clock window is a child of the notification area, also known as the taskbar tray. The taskbar tray's class name is "TrayNotifyWnd," which is itself a child of the taskbar, whose class name is "Shell_TrayWnd." To obtain the clock's window handle, work your way down this hierarchy, starting with the taskbar.

Use FindWindow only to retrieve the handle of the taskbar—a top-level window. From that level down, call FindWindowEx using the class names you uncovered with Spy++ (see Listing 3). FindWindowEx allows you to specify the handle of the window whose children are searched as its first parameter. If this parameter is Null, the search is among children of the desktop. The second parameter to FindWindowEx indicates that the search should begin with the next child window in ZOrder following that specified, allowing an iterative search through all children of any given parent. If this parameter is Null, the search begins with the first child window. The third and fourth parameters are the familiar class name and hWnd, as used with FindWindow—arguments that specify what criteria to use in the search.

Code Online

You can find all the code published in this issue of VBPJ on The Development Exchange (DevX) at http://www.windx.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

Who's My Parent?

Locator+ Codes

Listings for the entire issue (free Registered Level): VBPJ0598
Listings for this article only (subscriber Premier Level): AP0598

VB4 32-bit VB5

```
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 Form_Click()
  Call ShellLnk("c:\windows\start " & _
     "menu\programs\accessories\calculator.lnk")
End Sub

Private Function ShellLnk(ByVal LnkFile As String) As Long
  '
  ' Safe to ignore default directory and show mode
  ' in call to ShellExecute as those defined in the
  ' shortcut will take precedence.
  '
  ShellLnk = ShellExecute(0&, vbNullString, LnkFile, _
     vbNullString, vbNullString, vbNormalFocus)
End Function
```

LISTING 2 ***Shelling a Shortcut.*** *VB's Shell function doesn't work with LNK (shortcut) files, but that's no reason to think it can't be done. Calling the ShellExecute API function against a LNK file achieves the desired result. Don't worry about most of the parameters to ShellExecute, because the settings in the shortcut take precedence.*

VB4 32-bit VB5

```
Private Declare Function FindWindow Lib "user32" Alias _
  "FindWindowA" (ByVal lpClassName As String, ByVal _
  lpWindowName As String) As Long
Private Declare Function FindWindowEx Lib "user32" _
  Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal _
  hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 _
  As String) As Long
Private Declare Function GetDC Lib "user32" (ByVal _
  hwnd As Long) As Long
Private Declare Function FloodFill Lib "gdi32" (ByVal _
  hdc As Long, ByVal x As Long, ByVal y As Long, _
  ByVal crColor As Long) As Long
Private Declare Function ReleaseDC Lib "user32" _
  (ByVal hwnd As Long, ByVal hdc As Long) As Long

Private Sub Form_Click()
  Dim hClock As Long
  Dim hClockDC As Long
  Dim Buffer As String

  hClock = hTrayClock()
  hClockDC = GetDC(hClock)
  Call FloodFill(hClockDC, 1, 1, vbWhite)
  Call ReleaseDC(hClock, hClockDC)
End Sub

Private Function hTrayClock() As Long
  Dim hTaskbar As Long
  Dim hNotify As Long

  hTaskbar = FindWindow("Shell_TrayWnd", vbNullString)
  hNotify = FindWindowEx(hTaskbar, 0&, _
     "TrayNotifyWnd", vbNullString)
  hTrayClock = FindWindowEx(hNotify, 0&, _
     "TrayClockWClass", vbNullString)
End Function
```

LISTING 3 ***Drilling Down from a Top-Level Window.*** *FindWindow works great for top-level windows, but you must use FindWindowEx to retrieve the handle of child windows. This example drills down two levels to obtain the handle for the clock that appears in the taskbar notification area (tray), then fills it with the current brush.*