Handle Strings, Dates, and Colors

by Karl E. Peterson and Phil Weber

Click & Retrieve Source CODE!

Q PASSING STRINGS BETWEEN INSTANCES

I've successfully associated my application's default data-file extension with my application, so when a user double-clicks on a data file, my application automatically launches and opens the file. Now, I'd like to prevent a new instance of my application from starting each time the user double-clicks on a data file, because it's a multiple document interface (MDI) application. I know each new instance is invoked with the requested file name as a command-line parameter. How can I pass this information to a previous instance?

A

One of the best features of 32-bit Windows is that each application operates entirely within its own virtual memory space. This prevents any application from messing with the data of any other, as was possible under 16-bit Windows. However, sometimes simply passing a pointer to your own data space makes sense, or at least is the simplest way to share data. In this case, the system is designed to protect you from yourself.

One common workaround to sharing data between applications is using memory-mapped files [see L.J. Johnson's Windows Programming column, "The Persistence Of Memory," *VBPJ* July 1996]. This approach is best suited to large amounts of data, used repeatedly. For one-time passing of data between applications, Win32 introduced WM_COPYDATA, a new window message that uses the same memory-mapped file technique but hides the complexity behind a familiar interface. To use this technique, you need to write code for four tasks:

- Detecting and activating a previous application instance.
- Subclassing your main form and watching for the WM_COPYDATA message.
- Preparing the data for transmission.
- Making sense of the received data.

The first two items in this list have been covered in great detail over the years [see "Subclass Your Way Around VB's Limitations" by Karl E. Peterson and Jonathan Wood, *VBPJ* September 1995; and Mark Pruett's Programming Techniques column, "Who Says You Can't Write Custom Controls?" *VBPJ* July 1996). In fact, you can use the EnumWindows example I wrote about last month [Ask the VB Pro, *VBPJ* June 1998] to find a previous instance of your application, or you can take a more straightforward approach (see Listing 1). For simplicity, the sample code uses the freeware MsgHook control—available on my Web site at http://www.mvps.org/vb—for subclassing.

The WM_COPYDATA message passes a pointer to a COPYDATASTRUCT structure in lParam. The structure comprises three elements: the *dwData* element can be any 32-bit (Long) data you wish to pass; the *cbData* element specifies the size in bytes of the data you're passing; and the *lpData* element is a pointer to your data. The code in Listing 1 shows how to populate these elements using either ANSI or Unicode strings. VB's native string format is Unicode, so that approach is far simpler. Remember that Unicode uses two bytes per character, so be sure to use the LenB function when assigning *cbData*. Finally, assign *lpData* by using the undocumented StrPtr function, and call SendMessage.

If App.PrevInstance is True in your Form_Load event, call the SendMessageTo routine. Be sure the caption of your new form has been changed so it doesn't match the previous instance's caption. Pass the sought-after caption and the new command line (see Command$ in the help file) to SendMessageTo, then unload and exit the new instance.

By night, Karl E. Peterson is an independent programming consultant specializing in ActiveX controls. By day, he works as Geographical Information Systems analyst with a regional transportation planning agency. Karl coauthored Visual Basic 4 How-To, *from Waite Group Press. Online, he's a Microsoft MVP and a section leader in several* VBPJ *online forums. Find more of Karl's VB samples at http://www.mvps.org/vb.*

Phil Weber is an independent consultant specializing in Visual Basic and Web site development. He's a Microsoft Certified Solution Developer and Product Specialist. His contributions to this issue's column are subliminal. Find more of Phil's VB tips on his Web site at http://www.teleport.com/~pweber.

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, or ideas on the site, or access a comprehensive database of previously answered questions.

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  SetForegroundWindow Lib _
  "user32" (ByVal hWnd As Long) As Long
Private Declare Function IsIconic Lib  "user32" (ByVal _
  hWnd As Long) As Long
Private Declare Function ShowWindow Lib  "user32" _
  (ByVal hWnd As Long, ByVal nCmdShow As Long) As Long
Private Declare Function SendMessage Lib  "user32" _
  Alias "SendMessageA" (ByVal hWnd As Long, _
  ByVal wMsg As Long, ByVal wParam As Long, _
  Param As Any) As Long
' Required for VB4/32:
' Declare Function VarPtr Lib  "VB40032.DLL" (lpVar As
' Any) As Long

Private Const WM_COPYDATA = &H4A
Private Const SW_RESTORE = 9

Private Type COPYDATASTRUCT
  dwData As Long
  cbData As Long
  lpData As Long
End Type

Private Sub SendStringTo(ByVal Caption As String, _
  ByVal Send As String)
  Dim hWnd As Long
  Dim cds As COPYDATASTRUCT
  '
  ' Check for MDI form first, if no MDI form is found
  ' look for regular form. Use the classname just to
  ' narrow down the potential number of matching
  ' windows.
  '
  ' Use "ThunderRTMDIForm" for VB4
  hWnd = FindWindow ("ThunderRT5MDIForm", Caption)
  If hWnd = 0 Then
     ' Use "ThunderRTForm" for VB4
     hWnd = FindWindow ("ThunderRT5Form", Caption)
  End If
  '
  ' Activate window to which we're passing data.
  '
  If hWnd Then
     If IsIconic(hWnd) Then
        Call ShowWindow(hWnd,  SW_RESTORE)
     End If
     Call SetForegroundWindow(hWnd)
  Else
     Exit Sub 'no need to continue.
  End If

' ***  ANSI version -- more work!
'    Dim b() As Byte
'    b = StrConv(Send, vbFromUnicode)
'    cds.cbData = UBound(b) + 1
'    cds.lpData = VarPtr(b(0))
'    Call SendMessage(hWnd,  WM_COPYDATA, Me.hWnd, cds)
' ***  End ANSI implementation

' ***  UniCode version -- much cleaner!
  cds.cbData = LenB(Send)
  cds.lpData = StrPtr(Send)
  Call SendMessage(hWnd, WM_COPYDATA, Me.hWnd, cds)
' *** End Unicode implementation
End Sub
```

LISTING 1 ***Passing a String to Another VB App.*** *This routine demonstrates using WM_COPYDATA to pass string data from one VB application to another. Before calling SendMessage, the routine finds the other VB application and activates it with the help of a few API calls. You need less code by maintaining the string data as Unicode. VB4/32 users must use the ANSI version, adding a Declare for the VarPtr function.*

When your previous instance receives the WM_COPYDATA message, first extract the COPYDATASTRUCT structure from the memory location lParam points to (see Listing 2). One call to CopyMem (alias of RtlMoveMemory) accomplishes this task. Then, prepare a string buffer one-half the length specified in *cbData*, and again call CopyMem to sling the string data into your buffer. At this point, you're free to call your application's FileOpen routine or otherwise act on the received string.

WM_COPYDATA isn't limited to strings. Using this message, you can pass any sort of binary data you wish. Finally, be aware that App.PrevInstance testing doesn't work within the Integrated Development Environment (IDE), so this sort of code needs to be debugged as an EXE. Download a sample application on the free, Registered Level of The Development Exchange (see the Code Online box at the end of this column for details). —*K.E.P.*

VB4 32-bit VB5

```
Private Sub Msghook_Message(ByVal msg  As Long, ByVal wp As Long, _
  ByVal lp As Long, result As Long)

  Dim cds As COPYDATASTRUCT
  Dim Send As String

  If msg = WM_COPYDATA Then
     CopyMem cds, ByVal lp, Len(cds)

' ***  ANSI version -- more work!
'    Dim b() As Byte
'    ReDim b(0 To cds.cbData - 1) As Byte
'    CopyMem b(0), ByVal  cds.lpData, cds.cbData
'    Send = StrConv(b, vbUnicode)
' *** End ANSI implementation

' *** UniCode version -- much cleaner!
     Send = Space$(cds.cbData \ 2)
     CopyMem ByVal StrPtr(Send), ByVal cds.lpData, cds.cbData
' *** End Unicode implementation

     result = True
  End If
End Sub
```

LISTING 2 ***Receiving a String From Another VB App.*** *The freeware MsgHook control hooks the WM_COPYDATA message, allowing reception of passed strings. One call to CopyMem retrieves the COPYDATASTRUCT structure from memory into a local variable, and another CopyMem call extracts the string. Unfortunately, VB4/32 users must use the ANSI version.*

Q NEED FULL DATE PROPERTY SUPPORT

I'm writing a user control that requires setting several date properties. Some

of my users have complained they don't like to use the native Date data type, and they've asked me to find a more versatile method to expose these properties. How can I make these folks happy?

A New users of VB should be for-given for being unaware of the idiosyncrasies some of us old-timers still hold onto. Long before BASIC offered a Date data type (going back to the early days of QuickBasic), serial dates were stored in Double precision variables. Of course, some folks might also want to use strings as well, especially if users set the property directly.

I wrote a routine some time ago for handling just such a situation in one of my controls (see Listing 3). You can use the Value property Let/Get pair in either a class or a user control; the pair accepts and returns data using the Variant data type. This allows a user to pass just about anything, and it's up to the Let procedure to interpret what was passed.

Numerics must be validated to ensure they're within the valid date range supported by VB. The easiest way to do this is to turn on error trapping and attempt to convert the input to a date using


```
Private m_DateValue As Double

Public Property Let Value(ByVal  NewDate As Variant)
  '
  ' Attempt to make sense of input
  '
  If IsNumeric(NewDate) Then
     ' Error-trap for out of range
     On Error GoTo BadDateValue
        NewDate = CDate(NewDate)
     On Error GoTo 0
     m_DateValue = CDbl(NewDate)
  ElseIf IsNull(NewDate) = True Or  NewDate = "" Then
     ' Default to current date.
     m_DateValue = Now
  ElseIf IsDate(NewDate) Then
     ' Works for both String and Date types.
     m_DateValue =  CDbl(CDate(NewDate))
  Else
     GoTo BadDateValue
  End If
  PropertyChanged "Value"
  ' Update display (or whatever)
  Call SetDateValue
Exit Property

BadDateValue:
  Err.Raise Number:=errInvalidDate, Source:=ModuleName _
     & ".Value", Description:=msgInvalidDate
End Property

Public Property Get Value() As Variant
  '
  ' Return whole portion of stored date value, or Null
  ' if no stored value.
  '
  If m_DateTime Then
     Value = CDate(Fix(m_DateTime))
  Else
     Value = Null
  End If
End Property
```

LISTING 3 ***Providing Maximum Support for Date Properties.*** *This Let/Get pair of date-property procedures allows your users to pass any conceivable date representation to your class or user control. It's assumed you'll use the current date in place of Null input, but others could be added as well. You can handle Times similarly by using the fractional portion of the input only, rather than the whole part.*

VB5

```
Private Const defBackColor = vbWindowBackground
Private m_BackColor As Long

Private Sub UserControl_InitProperties()
  m_BackColor = defBackColor
End Sub

Private Sub UserControl_ReadProperties (PropBag As _
  PropertyBag)
  m_BackColor = PropBag.ReadProperty ("BackColor", _
     defBackColor)
End Sub

Private Sub UserControl_WriteProperties (PropBag As _
  PropertyBag)
  Call PropBag.WriteProperty ("BackColor", _
     m_BackColor, defBackColor)
End Sub

Public Property Let BackColor(ByVal NewVal As OLE_COLOR)
  m_BackColor = NewVal
  PropertyChanged "BackColor"
  Call SetColors
End Property

Public Property Get BackColor() As OLE_COLOR
  BackColor = m_BackColor
End Property

Public Function CheckSysColor(ByVal Color As Long) _
  As Long
  Const HighBit = &H80000000
  '
  ' If high bit set, strip, and get system color.
  '
  If Color And HighBit Then
     CheckSysColor = GetSysColor _
        (Color And Not HighBit)
  Else
     CheckSysColor = Color
  End If
End Function
```

LISTING 4 ***Providing Maximum Support for Color Properties.*** *This code illustrates all that's required for full support of color properties in your user controls. Using OLE_COLOR as the property type adds support for the drop-down color palette in the Properties window. Storing the value exactly as received from the user adds support for system-color constants. When using the API for painting, simply call CheckSysColor with the cached color value.*

CDate. If the input is either Null or an empty string, it's your call on how to handle it. In this case, my routine simply uses Now as the default value. The IsDate function validates string input. Interestingly, if a user passes a Date variable to the Value property, it too gets validated with the IsDate test. Any input data that fails these tests raises an error.

TO TURN ON COLOR PALETTE DROP-DOWNS, DECLARE YOUR COLOR PROPERTIES AS THE TYPE OLE_COLOR.

Because the Let procedure accepts a Variant, Get must also provide the cached property in this format. I prefer to always use Double variables to store dates and times, as this is the native format for VB's serial dates. Using this format, the date is stored in the whole portion and the time in the fractional portion of the variable. —*K.E.P.*

Q TURN ON FULL COLOR PROPERTY SUPPORT

How do I turn on the color palette drop-downs in the Properties window for my user control? Without these drop-downs, my work just doesn't seem professional.

A

Here's a simple trick for turning on those drop-downs. Declare your color properties as the type OLE_COLOR (see Listing 4). That's all there is to it. You beg to differ? Can I please explain all the extra code in that listing? Well, I admit, supporting this extra capability carries a *little* extra work when you actually use the colors your users assign.

VB toggles the high bit of a Long value to indicate that the low byte represents a system-color constant. Because your users have access to the system-color palette, you must be prepared to work with those values. You can translate a VB color constant into the actual RGB value by first turning off the high bit, then calling the GetSysColor API.

It's important you don't store these translated values in your control's PropertyBag, though. If you do, the control can't adopt to each user's system-color settings. Instead, call the CheckSysColor function, passing the cached color property value before any calls you might make to graphical API functions:

```
hBrush = CreateSolidBrush( _
   CheckSysColor(m_BackColor))
hPen = CreatePen(PS_SOLID, 0, _
   CheckSysColor(m_BackColor))
```

This approach is straightforward and gives your users the impression you're really on top of things. —*K.E.P.*

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.

Handle Strings, Dates, and Colors
Locator+ Codes

Listings for the entire issue, plus a demo that detects and activates a previous instance, should one exist (free Registered Level): VBPJ0798

Listings for this article only, the demo described above, plus the MDI sample that shipped with VB5, reworked to add support for previous instance detection and activation, as well as passing the command line to the previous instance (subscriber Premier Level): AP0798