Provide support for subclassing Unicode Windows

Summary

Id:	17.1
Type:	Issue
Current Status:	Open

Detail

14 Apr 2003 Open Matt Funnell

First of all, many thanks for the great code on your site - has been very, very useful.........

I came across a small problem when trying to subclass the internet explorer windows which seem to be Unicode windows. I've made some small changes to the recent code from your site and have attached below if interested.

Thanks again for your great site and your efforts.

```
' declares:
Private Declare Function IsWindow Lib "user32" ( _
    ByVal hWnd As Long) As Long
Private Declare Function IsWindowUnicode Lib "user32" ( _
    ByVal hWnd As Long) As Long

Private Declare Function GetProp Lib "user32" Alias "GetPropA" ( _
    ByVal hwnd As Long, ByVal lpString As String) As Long
Private Declare Function SetProp Lib "user32" Alias "SetPropA" ( _
    ByVal hwnd As Long, ByVal lpString As String, ByVal hData As Long) As Long
Private Declare Function RemoveProp Lib "user32" Alias "RemovePropA" ( _
    ByVal hwnd As Long, ByVal lpString As String) As Long
Private Declare Function GetPropW Lib "user32" ( _
    ByVal hwnd As Long, ByVal lpString As Long) As Long
Private Declare Function SetPropW Lib "user32" ( _
    ByVal hwnd As Long, ByVal lpString As Long, ByVal hData As Long) As Long
Private Declare Function RemovePropW Lib "user32" ( _
    ByVal hwnd As Long, ByVal lpString As Long) As Long

Private Declare Function CallWindowProc Lib "user32" _
Alias "CallWindowProcA" ( _
ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, _
ByVal Msg As Long, _
    ByVal wParam As Long, ByVal lParam As Long) As Long
Private Declare Function CallWindowProcW Lib "user32" ( _
ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, _
ByVal Msg As Long, _
    ByVal wParam As Long, ByVal lParam As Long) As Long

Private Declare Function SetWindowLong Lib "user32" _
Alias "SetWindowLongA" ( _
    ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Private Declare Function GetWindowLong Lib "user32" _
Alias "GetWindowLongA" ( _
    ByVal hwnd As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLongW Lib "user32" ( _
ByVal hwnd As Long, ByVal nIndex As Long, _
    ByVal dwNewLong As Long) As Long
Private Declare Function GetWindowLongW Lib "user32" ( _
    ByVal hwnd As Long, ByVal nIndex As Long) As Long

Private Declare Function GetWindowThreadProcessId Lib "user32" ( _
    ByVal hwnd As Long, lpdwProcessId As Long) As Long
Private Declare Function GetCurrentProcessId Lib "kernel32" () As Long

Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" ( _
    lpvDest As Any, lpvSource As Any, ByVal cbCopy As Long)

Private Const GWL_WNDPROC = (-4)
Private Const WM_DESTROY = &H2

' SubTimer is independent of VBCore, so it hard codes error handling

Public Enum EErrorWindowProc
    eeBaseWindowProc = 13080 ' WindowProc
    eeCantSubclass           ' Can't subclass window
    eeAlreadyAttached        ' Message already handled by another class
    eeInvalidWindow          ' Invalid window
    eeNoExternalWindow       ' Can't modify external window
End Enum

Private m_iCurrentMessage As Long
Private m_iProcOld As Long
Private m_f As Long


Public Property Get CurrentMessage() As Long
   CurrentMessage = m_iCurrentMessage
End Property

Private Sub ErrRaise(e As Long)
Dim sText As String, sSource As String
   If e > 1000 Then
      sSource = App.EXEName & ".WindowProc"
      Select Case e
      Case eeCantSubclass
         sText = "Can't subclass window"
      Case eeAlreadyAttached
         sText = "Message already handled by another class"
      Case eeInvalidWindow
         sText = "Invalid window"
      Case eeNoExternalWindow
         sText = "Can't modify external window"
      End Select
      Err.Raise e Or vbObjectError, sSource, sText
   Else
      ' Raise standard Visual Basic error
      Err.Raise e, sSource
   End If
End Sub

Private Function SetWindowProc(ByVal hWnd As Long, ByVal lpFn As Long) As Long
   If (IsWindowUnicode(hWnd) = 0) Then
      ' Not a Unicode window:
      SetWindowProc = SetWindowLong(hWnd, GWL_WNDPROC, AddressOf WindowProc)
   Else
      ' Unicode window:
      SetWindowProc = SetWindowLongW(hWnd, GWL_WNDPROC, AddressOf WindowProc)
   End If

End Function

Private Property Get MessageCount(ByVal hWnd As Long) As Long
Dim sName As String
   sName = "C" & hWnd
   MessageCount = GetProp(hWnd, sName)
End Property
Private Property Let MessageCount(ByVal hWnd As Long, ByVal count As Long)
Dim sName As String
Dim hData As Long
   m_f = 1
   sName = "C" & hWnd
   m_f = SetProp(hWnd, sName, count)
   If (count = 0) Then
```

```
      hData = RemoveProp(hWnd, sName)
   End If
   logMessage "Changed message count for " & Hex(hWnd) & " to " & count
End Property

Private Property Get OldWindowProc(ByVal hWnd As Long) As Long
Dim sName As String
   sName = hWnd
   OldWindowProc = GetProp(hWnd, sName)
End Property
Private Property Let OldWindowProc(ByVal hWnd As Long, ByVal lPtr As Long)
Dim sName As String
Dim hData As Long
   m_f = 1
   sName = hWnd
   m_f = SetProp(hWnd, sName, lPtr)
   If (lPtr = 0) Then
      hData = RemoveProp(hWnd, sName)
   End If
   logMessage "Changed Window Proc for " & Hex(hWnd) & " to " & Hex(lPtr)
End Property

Private Property Get MessageClassCount( _
    ByVal hWnd As Long, ByVal iMsg As Long) As Long
Dim sName As String
   sName = hWnd & "#" & iMsg & "C"
   MessageClassCount = GetProp(hWnd, sName)
End Property

Private Property Let MessageClassCount( _
    ByVal hWnd As Long, ByVal iMsg As Long, ByVal count As Long)
Dim sName As String
Dim hData As Long
   sName = hWnd & "#" & iMsg & "C"
   m_f = SetProp(hWnd, sName, count)
   If (count = 0) Then
      hData = RemoveProp(hWnd, sName)
   End If
   logMessage "Changed message count for " & Hex(hWnd) & _
        " Message " & iMsg & " to " & count
End Property

Private Property Get MessageClass( _
ByVal hWnd As Long, ByVal iMsg As Long, _
    ByVal index As Long) As Long
Dim sName As String
   sName = hWnd & "#" & iMsg & "#" & index
   MessageClass = GetProp(hWnd, sName)
End Property
Private Property Let MessageClass( _
ByVal hWnd As Long, ByVal iMsg As Long, _
    ByVal index As Long, ByVal classPtr As Long)
Dim sName As String
Dim hData As Long
   sName = hWnd & "#" & iMsg & "#" & index
   m_f = SetProp(hWnd, sName, classPtr)
   If (classPtr = 0) Then
      hData = RemoveProp(hWnd, sName)
   End If
   logMessage "Changed message class for " & Hex(hWnd) & _
     " Message " & iMsg & " Index " & index & " to " & Hex(classPtr)
End Property

Sub AttachMessage( _
      iwp As ISubclass, _
      ByVal hWnd As Long, _
      ByVal iMsg As Long _
   )
```

```
Dim procOld As Long
Dim msgCount As Long
Dim msgClassCount As Long
Dim msgClass As Long

   ' -------------------------------------------------------------------
   ' 1) Validate window
   ' -------------------------------------------------------------------
   If IsWindow(hWnd) = False Then
      ErrRaise eeInvalidWindow
      Exit Sub
   End If
   If IsWindowLocal(hWnd) = False Then
      ErrRaise eeNoExternalWindow
      Exit Sub
   End If

   ' -------------------------------------------------------------------
   ' 2) Check if this class is already attached for this message:
   ' -------------------------------------------------------------------
   msgClassCount = MessageClassCount(hWnd, iMsg)
   If (msgClassCount > 0) Then
      For msgClass = 1 To msgClassCount
         If (MessageClass(hWnd, iMsg, msgClass) = ObjPtr(iwp)) Then
            ErrRaise eeAlreadyAttached
            Exit Sub
         End If
      Next msgClass
   End If

   ' -------------------------------------------------------------------
   ' 3) Associate this class with this message for this window:
   ' -------------------------------------------------------------------
   MessageClassCount(hWnd, iMsg) = MessageClassCount(hWnd, iMsg) + 1
   If (m_f = 0) Then
      ' Failed, out of memory:
      ErrRaise 5
      Exit Sub
   End If

   ' -------------------------------------------------------------------
   ' 4) Associate the class pointer:
   ' -------------------------------------------------------------------
   MessageClass(hWnd, iMsg, MessageClassCount(hWnd, iMsg)) = ObjPtr(iwp)
   If (m_f = 0) Then
      ' Failed, out of memory:
      MessageClassCount(hWnd, iMsg) = MessageClassCount(hWnd, iMsg) - 1
      ErrRaise 5
      Exit Sub
   End If

   ' -------------------------------------------------------------------
   ' 5) Get the message count
   ' -------------------------------------------------------------------
   msgCount = MessageCount(hWnd)
   If msgCount = 0 Then

      ' Subclass window by installing window procedure
      procOld = SetWindowProc(hWnd, AddressOf WindowProc)
      If procOld = 0 Then
         ' remove class:
         MessageClass(hWnd, iMsg, MessageClassCount(hWnd, iMsg)) = 0
         ' remove class count:
         MessageClassCount(hWnd, iMsg) = MessageClassCount(hWnd, iMsg) - 1

         ErrRaise eeCantSubclass
         Exit Sub
      End If
      ' Associate old procedure with handle
      OldWindowProc(hWnd) = procOld
      If m_f = 0 Then
         ' SPM: Failed to VBSetProp, windows properties database problem.
         ' Has to be out of memory.

         ' Put the old window proc back again:
         SetWindowProc hWnd, procOld
         ' remove class:
         MessageClass(hWnd, iMsg, MessageClassCount(hWnd, iMsg)) = 0
         ' remove class count:
         MessageClassCount(hWnd, iMsg) = MessageClassCount(hWnd, iMsg) - 1

         ' Raise an error:
         ErrRaise 5
         Exit Sub
      End If
   End If


   ' Count this message
   MessageCount(hWnd) = MessageCount(hWnd) + 1
   If m_f = 0 Then
      ' SPM: Failed to set prop, windows properties database problem.
      ' Has to be out of memory

      ' remove class:
      MessageClass(hWnd, iMsg, MessageClassCount(hWnd, iMsg)) = 0
      ' remove class count contribution:
      MessageClassCount(hWnd, iMsg) = MessageClassCount(hWnd, iMsg) - 1

      ' If we haven't any messages on this window then remove the subclass:
      If (MessageCount(hWnd) = 0) Then
         ' put old window proc back again:
         procOld = OldWindowProc(hWnd)
         If Not (procOld = 0) Then
            SetWindowProc hWnd, procOld
            OldWindowProc(hWnd) = 0
         End If
      End If

      ' Raise the error:
      ErrRaise 5
      Exit Sub
   End If

End Sub

Sub DetachMessage( _
      iwp As ISubclass, _
      ByVal hWnd As Long, _
      ByVal iMsg As Long _
   )
Dim msgClassCount As Long
Dim msgClass As Long
Dim msgClassIndex As Long
Dim msgCount As Long
Dim procOld As Long

   ' --------------------------------------------------------------------
   ' 1) Validate window
   ' --------------------------------------------------------------------
   If IsWindow(hWnd) = False Then
      ' for compatibility with the old version, we don't
      ' raise a message:
      ' ErrRaise eeInvalidWindow
      Exit Sub
   End If
   If IsWindowLocal(hWnd) = False Then
      ' for compatibility with the old version, we don't
      ' raise a message:
      ' ErrRaise eeNoExternalWindow
      Exit Sub
   End If

   ' -------------------------------------------------------------------
   ' 2) Check if this message is attached for this class:
   ' -------------------------------------------------------------------
   msgClassCount = MessageClassCount(hWnd, iMsg)
   If (msgClassCount > 0) Then
      msgClassIndex = 0
      For msgClass = 1 To msgClassCount
         If (MessageClass(hWnd, iMsg, msgClass) = ObjPtr(iwp)) Then
            msgClassIndex = msgClass
            Exit For
         End If
      Next msgClass

      If (msgClassIndex = 0) Then
         ' fail silently
         Exit Sub
      Else
         ' remove this message class:

         ' a) Anything above this index has to be shifted up:
         For msgClass = msgClassIndex To msgClassCount - 1
            MessageClass(hWnd, iMsg, msgClass) = MessageClass(hWnd, iMsg, msgClass + 1)
         Next msgClass

         ' b) The message class at the end can be removed:
         MessageClass(hWnd, iMsg, msgClassCount) = 0

         ' c) Reduce the message class count:
         MessageClassCount(hWnd, iMsg) = MessageClassCount(hWnd, iMsg) - 1

      End If

   Else
      ' fail silently
      Exit Sub
   End If

   ' --------------------------------------------------------------------
   ' 3) Reduce the message count:
   ' --------------------------------------------------------------------
   msgCount = MessageCount(hWnd)
   If (msgCount = 1) Then
      ' remove the subclass:
      procOld = OldWindowProc(hWnd)
      If Not (procOld = 0) Then
         ' Unsubclass by reassigning old window procedure
         SetWindowProc hWnd, procOld
      End If
      ' remove the old window proc:
      OldWindowProc(hWnd) = 0
   End If
   MessageCount(hWnd) = MessageCount(hWnd) - 1

End Sub

Private Function WindowProc( _
      ByVal hWnd As Long, _
      ByVal iMsg As Long, _
      ByVal wParam As Long, _
      ByVal lParam As Long _
   ) As Long

Dim procOld As Long
Dim msgClassCount As Long
Dim bCalled As Boolean
Dim pSubClass As Long
Dim iwp As ISubclass
Dim iwpT As ISubclass
Dim iIndex As Long
Dim bDestroy As Boolean

   ' Get the old procedure from the window
   procOld = OldWindowProc(hWnd)
   Debug.Assert procOld <> 0

   If (procOld = 0) Then
      ' we can't work, we're not subclassed properly.
      Exit Function
   End If

   ' SPM - in this version I am allowing more than one class to
   ' make a subclass to the same hWnd and Msg.  Why am I doing
   ' this?  Well say the class in question is a control, and it
   ' wants to subclass its container.  In this case, we want
   ' all instances of the control on the form to receive the
   ' form notification message.

   ' Get the number of instances for this msg/hwnd:
   bCalled = False

   If (MessageClassCount(hWnd, iMsg) > 0) Then
      iIndex = MessageClassCount(hWnd, iMsg)

      Do While (iIndex >= 1)
         pSubClass = MessageClass(hWnd, iMsg, iIndex)

         If (pSubClass = 0) Then
            ' Not handled by this instance
         Else
            ' Turn pointer into a reference:
            CopyMemory iwpT, pSubClass, 4
            Set iwp = iwpT
            CopyMemory iwpT, 0&, 4

            ' Store the current message, so the client can check it:
            m_iCurrentMessage = iMsg

            With iwp
               ' Preprocess (only checked first time around):
               If (iIndex = 1) Then
                  If (.MsgResponse = emrPreprocess) Then
                     If Not (bCalled) Then
                        If (IsWindowUnicode(hWnd) = 0) Then
                           WindowProc = CallWindowProc(procOld, hWnd, iMsg, _
                                            wParam, ByVal lParam)
                        Else
                           WindowProc = CallWindowProcW(procOld, hWnd, iMsg, _
                                            wParam, ByVal lParam)
                        End If
                        bCalled = True
                     End If
                  End If
               End If
               ' Consume (this message is always passed to all control
               ' instances regardless of whether any single one of them
               ' requests to consume it):
               WindowProc = .WindowProc(hWnd, iMsg, wParam, ByVal lParam)
            End With
         End If

         iIndex = iIndex - 1
      Loop

      ' PostProcess (only check this the last time around):
      If Not (iwp Is Nothing) And Not (procOld = 0) Then
         If iwp.MsgResponse = emrPostProcess Then
            If Not (bCalled) Then
               If (IsWindowUnicode(hWnd) = 0) Then
                  WindowProc = CallWindowProc(procOld, hWnd, iMsg, _
                                       wParam, ByVal lParam)
               Else
                  WindowProc = CallWindowProcW(procOld, hWnd, iMsg, _
                                       wParam, ByVal lParam)
               End If
               bCalled = True
            End If
         End If
      End If

   Else
      ' Not handled:
      If (iMsg = WM_DESTROY) Then
         ' If WM_DESTROY isn't handled already, we should
         ' clear up any subclass
         pClearUp hWnd
         If (IsWindowUnicode(hWnd) = 0) Then
            WindowProc = CallWindowProc(procOld, hWnd, iMsg, _
                                 wParam, ByVal lParam)
         Else
            WindowProc = CallWindowProcW(procOld, hWnd, iMsg, _
                                 wParam, ByVal lParam)
         End If
      Else
         If (IsWindowUnicode(hWnd) = 0) Then
            WindowProc = CallWindowProc(procOld, hWnd, iMsg, _
                                 wParam, ByVal lParam)
         Else
            WindowProc = CallWindowProcW(procOld, hWnd, iMsg, _
                                 wParam, ByVal lParam)
         End If
      End If
   End If

End Function
Public Function CallOldWindowProc( _
      ByVal hWnd As Long, _
      ByVal iMsg As Long, _
      ByVal wParam As Long, _
      ByVal lParam As Long _
   ) As Long
Dim iProcOld As Long
   iProcOld = OldWindowProc(hWnd)
   If Not (iProcOld = 0) Then
      CallOldWindowProc = CallWindowProc(iProcOld, hWnd, iMsg, wParam, lParam)
   End If
End Function

Function IsWindowLocal(ByVal hWnd As Long) As Boolean
    Dim idWnd As Long
    Call GetWindowThreadProcessId(hWnd, idWnd)
    IsWindowLocal = (idWnd = GetCurrentProcessId())
End Function

Private Sub logMessage(ByVal sMsg As String)
   Debug.Print sMsg
End Sub
```

```
Private Sub pClearUp(ByVal hWnd As Long)
Dim msgCount As Long
Dim procOld As Long
   ' this is only called if you haven't explicitly cleared up
   ' your subclass from the caller.  You will get a minor
   ' resource leak as it does not clear up any message
   ' specific properties.
   msgCount = MessageCount(hWnd)
   If (msgCount > 0) Then
      ' remove the subclass:
      procOld = OldWindowProc(hWnd)
      If Not (procOld = 0) Then
         ' Unsubclass by reassigning old window procedure
         SetWindowProc hWnd, procOld
      End If
      ' remove the old window proc:
      OldWindowProc(hWnd) = 0
      MessageCount(hWnd) = 0
   End If
End Sub
```