Correcting HTML Help-based "What's This" Help in Visual Basic 6

Updated February 26, 2003

David E. Liske
Delmar Computing Services, Tecumseh, Michigan
http://www.mvps.org/htmlhelpcenter

Introduction

One interesting result of the research for the HTML Help class module for Visual Basic is that it fixes an irritating problem with using HTML Help-based What's This topics in Visual Basic 6 with versions of hhctrl.ocx earlier than 4.74.8637 *. This is where the popup appears behind the application, and the only indication that the popup has been created is a blank button on the taskbar. Clicking the button on the taskbar brings the popup to the front where it can be read. Testing has proven that using the technique described in the class module's documentation for the HHSubClass procedure fixes this problem. The popups subsequently appear in front of the application 100% of the time. The topic you're now reading is a description of how this is accomplished. The description here covers the subject with less detail than what's in the documentation for the class module itself, as this only describes what's needed and leaves out some additional functionality.

This technique for displaying What's This topics works quite well from Visual Basic 5 as well, which has no native method for displaying HTML Help-based What's This Help.

The Sample Application

If you'll download the source code for the sample application from the page you downloaded this paper from, you'll see that it illustrates both the problem and the solution. Starting the application from within Visual Basic displays form1.frm. Using the What's This function from the title bar button or the F1 key may, or may not, display the popup for the button in front of the form. Subsequent usage with the same instance of the form will show the popup appearing behind the form the majority of the time.

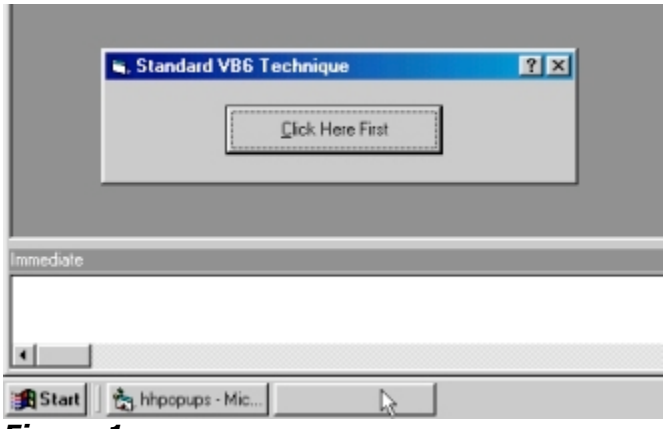

Figure 1
Clicking on the blank button on the taskbar will bring the popup to the front.

Another annoyance with this is that the popup is not attached to the form. From within the Visual Basic IDE, you can close the form, and actually close the entire application, and the button will still be displayed on the taskbar. Clicking the button will cause the popup to appear long after the application is gone.

Clicking the button itself will open form2.frm. Initializing What's This Help from this form numerous times will cause the popup to appear in front of the application without fail.

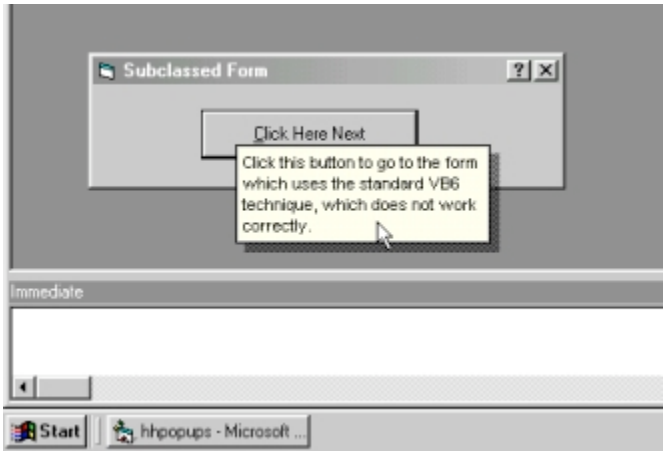

Figure 2
The subclassing method works correctly.

Form2.frm is displaying the popup topics in the same manner as is done with most other development packages. Once the form is subclassed, it intercepts the WM_HELP message and calls the HTML Help API HH_DISPLAY_TEXT_POPUP command via some properties and a method of the HTML Help class module, which is included in the sample application. The popup is displayed using the form's hWnd, so there's no chance of it hanging around to pop up and annoy your users later.

Using other portions of the HTML Help class module, the OnHelp procedure in the module for the subclassed form can be used to call other parts of the HTML Help API. This means an F1 keypress can be made to display almost any part of an HTML Help file. You may want to experiment with this in your own applications.

The Technique

The subclassing begins by having the form call the HHSubclass procedure included in the code module.

```
Private Sub Form_Load()

  HHSubclass Me

End Sub
```

The HHSubclass procedure uses the AddressOf operator to replace the form's standard procedure with the HHSubclassWndProc procedure. The form then gets added to a subclass collection.

```
' Subclassing
Private Const GWL_WNDPROC = (-4)
```

```
Private Declare Function GetWindowLong Lib "user32" _
Alias "GetWindowLongA" (ByVal hwnd As Long, _
ByVal nIndex 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 colHTMLHelp As New Collection

Public Sub HHSubclass(frm As Object)

' Uncomment this line in Debug mode (see the
' "Attention" section of the comment block for
' this module):
' Exit Sub

Dim hHelp As New HTMLHelp

' Create the object as a form
Set hHelp.frm = frm
hHelp.hwnd = frm.hwnd
hHelp.lpPrevWndFunc = GetWindowLong _
(frm.hwnd, _
GWL_WNDPROC)

' Replace the basic window procedure of the
' form calling this procedure
Call SetWindowLong(frm.hwnd, _
GWL_WNDPROC, _
AddressOf HHSubclassWndProc)

' Put this form into the subclass collection
' we created in the Declarations section
colHTMLHelp.Add hHelp

End Sub
```

The HHSubclassWndProc procedure looks at the Windows messages as they pass through. If a message is the WM_HELP message, the procedure passes the information from the message to an OnHelp procedure for the window the message belongs to. If it's not the WM_HELP message, the procedure sends the message on its way.

If the window doesn't exist any longer for some reason, we'll remove it from the collection.

```
Private Function HHSubclassWndProc(ByVal hwnd As Long, _
ByVal msgWinMessage As Long, ByVal wParam As Long, _
ByVal lParam As Long) As Long

Dim colhHelp As Object

' Loop through all the forms in the collection and use
' the handle to determing the message the form belongs to
For Each colhHelp In colHTMLHelp
```

```
If (colhHelp.hwnd = hwnd) Then
Exit For
End If
Next colhHelp

' Track down which message was sent and run the
' appropriate procedure on the calling form
Select Case (msgWinMessage)

Case WM_HELP
' The WM_HELP message is sent whenever the user
' presses the F1 key. It also occurs in response
' to What's This Help requests.
Dim hlpHelpInfo As HELPINFO
Call CopyMemory(hlpHelpInfo, ByVal lParam, Len(hlpHelpInfo))
Call colhHelp.frm.OnHelp(hlpHelpInfo.hItemHandle)

Case Else
' Let the message continue on its way
HHSubclassWndProc = CallWindowProc _
(colhHelp.lpPrevWndFunc, hwnd, msgWinMessage, _
wParam, ByVal lParam)

End Select

If (msgWinMessage = WM_NCDESTROY) Then
' If the window no longer exists,
' get it out of the HHSubclass collection
Dim intCount As Integer
For intCount = 1 To colHTMLHelp.Count
If (colhHelp Is colHTMLHelp(intCount)) Then
Call colHTMLHelp.Remove(intCount)
Exit For
End If
Next intCount
End If

End Function
```

The OnHelp procedure in the module for the subclassed form calls the properties and method that will create the popup. First, we declare some variables.

```
Public Sub OnHelp(hWndControl As Long)

On Error GoTo ErrHandler

Dim frmCurrent As Form
Dim ctlControl As Control
Dim errNumber As Integer
Dim strCustom As String
```

We then look at each control on the form to determine which one the popup might be for. If we find the right one, we start creating the popup from a CHM topic. Note that the popup could also come from a text string or a resource file.

```
For Each frmCurrent In Forms
```

```
For Each ctlControl In frmCurrent.Controls
If (ctlControl.hwnd = hWndControl) Then
With hHelp

' Set the popup type to CHM-based
.HHPopupType = HH_CHM_POPUP
```

The CHMFile and HHPopupFile properties for the class contain the names of the two files we need. This gives us the equivalent to what's in the Form_Load procedure in form1.frm.

```
' Set up the form to use HTML Help popups
App.HelpFile = App.Path & "\hhpopups.chm::/popups.txt"
```

We can then set the text color and size, and the popup color, to match standard popups.

```
' Specify the CHM file and the internal
' text popup file
.CHMFile = .HHSetHelpFile(1)
.HHPopupFile = .HHSetHelpFile(2)

' Set the colors and text to match those
' from a HH_TP_HELP_WM_HELP popup
.HHPopupCustomColors = True
.HHPopupCustomBackColor = &HFFFF
.HHPopupCustomTextColor = &HFFFF
.HHPopupTextSize = "8"
```

Next, we look at the Tag property for the control. This is where we store the context integer for the What's This popup. If the Tag property is empty, we can send up a generic "topic not found" topic. Note that this text exists in the code, not in the CHM.

```
' Get the context integer from the Tag property
.HHPopupID = CLng(ctlControl.Tag)

' If the control has a Tag property, we're ok
If errNumber <> 438 Then
' Tag property is empty

If ctlControl.Tag = "" Then
' No topic ID is specified in the Tag property,
' so send up a generic message in order to
' prevent HH error message.
strCustom = "HHSubclass message:" & _
Chr(10) & _
"You need to yell at the Help author for " & _
"not creating a Help topic for this item!"
.HHPopupText = strCustom
.HHPopupType = HH_TEXT_POPUP
.HHDisplayPopup Me.hwnd
```

Otherwise, we'll go ahead and display the popup.

```
Else
' Display the specified CHM-based popup topic
.HHDisplayPopup Me.hwnd
errNumber = 0
```

```
End If
```

If there's no Tag property, it needs to be dealt with. We do this in both the error handler, and within the procedure. This takes care of a couple of different situations whereby an error will be generated.

```
' If the control doesn't have a Tag property,
' deal with it.
Else
errNumber = 0
End If

End With
End If

Next ctlControl
Next frmCurrent

Exit Sub

ErrHandler:

Select Case Err.Number
Case 438
' The control being checked doesn't have a
' Tag property, so set the variable to Err.Number
' before it gets cleared
errNumber = Err.Number
Resume Next

Case Else
' Nothing else to really be concerned with
Resume Next

End Select

End Sub
```

Finally, closing the form calls the HHUnSubclass procedure prior to the form itself closing. This prevents Visual Basic from crashing.

```
Private Sub Form_Unload(Cancel As Integer)

HHUnSubClass Me

End Sub

Public Sub HHUnSubClass(frm As Object)

Dim hHelp As New HTMLHelp

' Release the subclassed form
Call SetWindowLong(frm.hwnd, _
GWL_WNDPROC, hHelp.lpPrevWndFunc)

End Sub
```

Summary

The native method for displaying HTML Help-based What's This popups in Visual Basic 6 has some major problems. Subclassing a form and intercepting the WM_HELP message is an accurate method for displaying popups consistently from HTML Help files in Visual Basic 6. It also works quite well with Visual Basic 5.

** 4.74.8637 is the version of HTML Help 1.3 hhctrl.ocx that ships with the Internet Explorer 5.5 Platform Preview. The final hhctrl.ocx build is 4.74.8702. Both of these fix the popup problem. Subclassing will still be necessary when using popups with Visual Basic 5 (since VB5 doesn't do HTML Help popups natively), but it will no longer be necessary in order to fix the popup problem with Visual Basic 6.*