Connecting WinHelp to Visual Basic Programs

by Don Lammers

Last updated 2000-06-01

Presented with author's permission by

Shadow Mountain Tech

Contents

Scope

Visual Basic provides several hooks for easily connecting to WinHelp. In addition, you can call the WinHelp API from anywhere in Visual Basic for additional access to help.

This document is intended to show you how to hook WinHelp to Visual Basic 5 and later *without* using add-ons.

Acknowledgements

The following information is from my own experimentation and from people who have walked this path before me. Thanks to all of the following:

David Liske, Dana Cline, Microsoft Help WorkShop Help, Microsoft Knowledge Base, *The Developer's Guide to WINHELP.EXE* (Jim Mischel), *Building Windows 95 Help* (Nancy Hickman), Paul O'Rear, *Developing Online Help for Windows 95* (Boggan, Farkas, and Welinske), Gordon F. MacLeod, Burt Abreu.

Connecting Context Sensitive WinHelp to Visual Basic

Visual Basic (versions 5 and later) provides directly for calling window level WinHelp topics using the F1 key, **or** providing What's This? Help for each control, including activation from the F1 key. The choice between What's This? Help and form level F1 help can be made on a form by form basis.

This section shows the steps necessary to hook context sensitive WinHelp into Visual Basic.

Set the Help File Name

Before using the built in context sensitive help features of Visual Basic, you need to set the App.HelpFile property so that the program knows the help file to call and the window in which WinHelp should display the topic.

To set the help file name:

- Set the App.HelpFile property. The basic syntax is:

  ```
  App.HelpFile = App.Path & "\helpfile.hlp>hWindow"
  ```

  helpfile.hlp is the name of the WinHelp file.

  hWindow is the name of the secondary window in which to display the topics. If not specified, the topic is displayed in the main window.

Tips:

- If you are calling both secondary windows and pop-ups from your application and only have a single help file, you may want to skip the secondary window name in this property and append it to App.HelpFile only when you call help that requires a secondary window.
- The App.HelpFile property can be changed at runtime, so you can reset it in code as needed to call additional help files.
- If you have more than one help file or window style to call from the program you should put the code for changing help file names and windows in a separate subroutine. This makes any changes during the project much easier to implement.
- You can let the help author edit the help file name and window specification by putting them in a separate text file. Setting this file up in INI file format lets you use standard INI file commands to access this "text database."

Make Sure WinHelp Can Find the Help File

Make sure that WinHelp can always find the help file. Since your program can be started from shortcuts created by the user, the installation program should register the help file or you should call help from your program using the full path and file name.

To ensure that the program and WinHelp can always find the help file:

- Register the help file by creating a string value under HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Help. The name of the string value is the help file name (without path) and the value is the path.
- Always use the full path and file name when setting App.HelpFile, even if the help file is in the same folder as the program executable.

Connect the F1 key

Visual basic uses the F1 key as one means to trigger help. For each form in your project, you can connect either form level or control level help to the F1 key, or turn the F1 key off completely.

To disconnect F1 help from a form:

1. Set the form's WhatsThisHelp property to False. On forms that support the WhatsThisButton property it will also be set to False. MDI forms do not support the WhatsThisButton property.
2. Set the form's HelpContextID property to 0 (zero).
3. Set the control's WhatsThisHelpID property to 0 (zero) for all controls in the form.

To connect form level help to a form:

1. Set the form's WhatsThisHelp property to False. On forms that support the WhatsThisButton property it will also be set to False. MDI forms do not support the WhatsThisButton property.
2. Set the form's HelpContextID property. This number must match the topic ID in the [MAP] section of the WinHelp project (.hpj) file. The topic will be opened in the window specified in App.HelpFile, or in the default window.

To connect control level help (also called What's This? Help) to a form:

1. Set the form's WhatsThisHelp property to True.
2. For each control with a What's This? Help topic, set the WhatsThisHelpID. This number must match the topic ID in the [MAP] section of the WinHelp project (.hpj) file.
3. To turn off What's This? Help for a single control, set the control's WhatsThisHelpID to 0 (zero).

Notes:

- Once you put a topic number in the WhatsThisHelpID property of a control, the KeyDown, KeyUp, and KeyPress events of the control are no longer triggered by the F1 key. You can detect any other key press in these events, but not F1.
- The WhatsThisHelp property for a form cannot be changed at runtime. The only way to turn What's This? Help on or off at run time is to set and clear the WhatsThisHelpID for the control.

Activate the What's This? Help Button in a Dialog Box Title Bar

If the form is not resizeable, you can put a ? in the title bar to activate What's This? Help mode. If the form is resizeable you must handle this function using a separate button. See *Connect What's This? Help to Resizeable Windows*.

To activate the What's This? Help button in the title bar:

1. Set the form's BorderStyle property to Fixed Single or Fixed Dialog.
2. Set the form's WhatsThisButton property to True. This also sets the WhatsThisHelp property to True.
3. For each control with a What's This? Help topic, set the WhatsThisHelpID. This number must match the topic ID in the [MAP] section of the WinHelp project (.hpj) file.

Note:

- You cannot have a What's This? Help button in the title bar of any resizeable window, or any window set to the Fixed ToolWindow style.

Connect What's This? Help to Resizeable Windows

The following is only necessary to activate What's This? Help in a resizeable window or in a window set to the Fixed ToolWindow style. Otherwise if the form is not resizeable you can have Visual Basic put a ? in the title bar to activate What's This? Help mode (see *Activate the What's This? Help Button in a Dialog Box Title Bar*).

1. Set the form's WhatsThisHelp property to True.
2. Create a What's This? Help button for the toolbar that can be toggled.
3. In the MouseDown event of What's This? Help button, place any code necessary for putting the button in it's "down" state.
4. In the MouseUp or Click event of What's This? Help button, place the following code: Visual Basic does not correctly trigger What's This? Help mode if you put this code in the MouseDown event.

```
'**Set What's This? Help mode
WhatsThisMode
'**Toggle the button state if necessary.
'**Any commands placed here will not be executed until
'**What's This? Help mode is released.
....Code for setting the What's This? Help button to "up"
```

   Notes:

   - This activates What's This? Help mode for visible child forms, but not for any other non-modal forms that may be open (for instance, tool windows).
   - Any code in the subroutine after you call the WhatsThisMode method will not be executed until What's This? Help mode is released by Visual Basic.
   - The next mouse down event will be processed by the What's This? Help logic without triggering an event you can intercept.
   - If you put this code in the Click event you can call the button Click subroutine from the What's This? Help menu item without worrying about parameters.
5. For each control with a What's This? Help topic, set the WhatsThisHelpID. This number must match the topic ID in the [MAP] section of the WinHelp project (.hpj) file.
6. In the Help menu, create an item called What's This? Help and assign SHIFT-F1 as the shortcut. In the Click event for this menu item, call the What's This? button Click or Mouse Up event where you put the WhatsThisMode command.
7. In the MouseUp or Click event of What's This? Help button, activate What's This? Help mode for any non-child non-modal form (such as a tool) that you want included in the What's This? Help logic. These forms must have a WhatsThisHelpID set for each control, and their WhatsThisHelp property set to True. The code would look something like this:

```
'**Set What's This? Help mode for tools that are open
If mnu_ColorWheel.Checked = True then
  FrmTool1.WhatsThisMode
End If
```

Tips:

- You can trigger What's This? Help mode programmatically from any control that offers a MouseUp or Click event.
- In C++, a menu item and equivalent toolbar item usually have the same control ID and thus the same context ID (though this can be overridden). In Visual Basic you can easily set the WhatsThisHelpID for each of these separately.

Implement F1 "Selected Text" Help

In any control where the user can highlight text, you can use this text to call an appropriate help topic. Because What's This? Help and F1 help are so closely tied together in Visual Basic, you cannot use F1 in this matter and still have an active What's This? Help button without additional programming. In many cases this kind of help is provided in child windows which would not have other forms of context sensitive help anyway. For instance, in Visual Studio Microsoft does not provide What's This? Help for the main windows but does provide "selected text" help. Mixing methods is discussed in the next section.

1. Set the control's WhatsThisHelpID property to 0 (zero).
2. Intercept the F1 KeyDown event. To do this, put the following code in the KeyDown event for the control:

```
'**Check for no highlighting
If frmChild1.Text1.selText = "" Then
  '**Nothing highlighted. We don't do anything.
  KeyCode = 0
ElseIf KeyCode = vbKeyF1 And Shift = 0 Then
  Dim selText$
  '**Strip spaces (if any)
  selText$ = Trim$(frmChild1.Text1.selText)
  ...any other code for determining the highlighted words
  '**Force the help file open to insure that the
  '**HELP_COMMAND command always work.
  '**The help file is forced open to the default topic,
  '**which is the "Keyword Not Found" topic, in the
  '**same secondary window that is normally used.
  success = WinHelp(hwnd, App.HelpFile, HELP_FORCEFILE, ByVal 0&)
  '**Set up the string for calling the KLink macro. We first need
  '**to use the TestKLink macro to see if there is a matching
  '**keyword. Note that the help compiler converts KLink to KL
  '**TestKLink to KL with the test parameter. Since this string
  '**is not going throught the help compiler, we need use the
  '**short form of the macro.
  HelpCommand$ = "IF(KL(`" & selText$ & "', 4, `', `'),`KL(`" &
selText$ & "', 1, `', `')')"
  '**Call the TestKLink macro and KLink macros.
  success = WinHelp(hwnd, App.HelpFile, HELP_COMMAND, ByVal
HelpCommand$)
else
  ...any other key trapping here
End If
```

Mixing What's This? Help with F1 "Selected Text" Help

Mixing standard What's This? Help with "selected text" help requires a bit of additional programming. The main reason is that the WhatsThisHelp property is cannot be changed at run time. To disconnect F1 help from What's This help requires that the WhatsThisHelpIDs for controls on a form be set to zero except when the form is in What's This? Help mode.

1. Create a subroutine to store the WhatsThisHelpID property for each control in its Tag property on startup.

```
Public Sub LoadTagsWithContextID(thisForm as Form)
  '**This just makes sure that the Tag property and the WhatsThisHelpID
  '**property are synchronized.
  For Each Control In thisForm.Controls
    Select Case LCase$(TypeName(Control))
    Case "commandbutton", "image", "picturebox"
      Control.Tag = Format(Control.WhatsThisHelpID, "General Number")
    Case "menu"
    End Select
  Next Control
End Sub
```

2. Create a subroutine to set and clear the What's This? Help status for the form:

```
Public Sub SetWhatsThisStatus(thisForm As Form, status%)
  '**This subroutine does the dirty work of copying the values
  '**from the Tag property ot erasing the WhatsThisHelpID.
  Dim fControls As Control
  Select Case status%
  Case 0
    '**No longer in What's This? Help. Erase all IDs
    For Each Control In thisForm.Controls
      Select Case LCase$(TypeName(Control))
      Case "commandbutton", "image", "picturebox", "textbox"
        Control.WhatsThisHelpID = 0
      Case "menu"
      End Select
    Next Control
    useWhatsThisHelp% = 0
  Case Else
  '**Going into What's This? Help. Copy IDs from Tag property
    For Each Control In thisForm.Controls
      Select Case LCase$(TypeName(Control))
      Case "commandbutton", "image", "picturebox", "textbox"
        Control.WhatsThisHelpID = Val(Control.Tag)
      Case "menu"
      End Select
    Next Control
    useWhatsThisHelp% = 1
  End Select
End Sub
```

3. Activate What's This? Help mode.

   For this example we will assume that the button will handle its own state change on MouseDown and MouseUp. If you are using a button without separate bitmaps for the MouseDown and MouseUp states, you will also have to add code to switch the button bitmap appropriately.

```
Private Sub btn_WhatsThisHelp_Click()
   '**Set WhatsThisHelpID values from the Tag property.
   Call SetWhatsThisStatus(frmMain, 1)
   Call SetWhatsThisStatus(frmChild1, 1)
   '**Set What's This? Help mode.
   WhatsThisMode
   '**Reset the WhatsThisHelp IDs to 0 (zero). These commands will
   '**not be activated until WhatsThisHelp mode is cleared.
   Call SetWhatsThisStatus(frmMain, 0)
   Call SetWhatsThisStatus(frmChild1, 0)
End Sub
```

Connect Context Help to Menu Items

When you create each menu item, include the context ID in the Menu Editor's HelpContextID box. When there is a context ID in this box, Visual Basic will open that topic in the secondary help window specified by App.HelpFile when you highlight the menu item and press F1.

Connect Context Help to a Right Click on a Button

The traditional way of connecting help to a right click on a control is to provide a pop-up menu with an item named What's This?. When the user clicks this menu item, help opens to the appropriate topic. the basic steps in creating such a link are:

1. In the control's Click event, save the topic number to call.
2. Implement whatever code is needed to open the pop-up menu with the appropriate items listed.
3. In the Click event for the What's This? menu item, call the WinHelp API with the appropriate topic number.

Tip:

- Unless there are only a few controls that do not implement a right click menu, I recommend calling the topic directly if the user right clicks a button that does not otherwise have a right click menu. A pop-up menu with a single What's This? item looks a bit strange, and this saves the user one click.

Calling WinHelp Directly from Your Visual Basic Code

You can programmatically call help from anywhere in your code. The most common reasons for doing this would be to open the help Table of Contents from the Help menu or to call a general help topic from a Help button on a form, but you can also call help using keywords.

WinHelp API Syntax

The **WinHelp** function starts Windows Help (WINHLP32.EXE) and passes additional data indicating the nature of the help requested by the application. To use the following API calls you must attach WinHelp.bas or Winhelp.cls (see *References* for where to get these).

```
BOOL WinHelp( HWND hWndMain, LPCTSTR lpszHelp,
     UINT uCommand, DWORD dwData )
```

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

hWndMain is the handle of the window requesting help. The **WinHelp** function uses this handle to keep track of which applications have requested help. If the *uCommand* parameter specifies HELP_CONTEXTMENU or HELP_WM_HELP, *hWndMain* identifies the control requesting help.

lpszHelp is a string containing the path, if necessary, and the name of the help file that WinHelp is to display.

Note: The filename may be followed by an angle bracket (>) and the name of a secondary window if the topic is to be displayed in a secondary window rather than in the primary window. The name of the secondary window must have been defined in the [WINDOWS] section of the help project (.hpj) file.

uCommand specifies the type of help requested. For a list of possible values and how they affect the value to place in the *dwData* parameter. For a partial list of possible values, see *Help Command Constants* below. For a full list, see the *Microsoft Help Workshop Help*.

dwData specifies additional data. For a partial list of possible values, see *Help Command Constants* below. For a full list, see the *Microsoft Help Workshop Help*.

WinHelp Command Constants

The following list shows the possible values for the *uCommand* parameter (WinHelp API) or the Common Dialog control HelpCommand property (CD), its value, its action, and the corresponding formats of the *dwDat*a command:

HELP_COMMAND (API) or cdlHelpCommand (CD) (258 or &H102) executes a WinHelp macro or macros. *dwData* is a string containing the WinHelp macro(s) to run. If the string specifies multiple macro names the names must be separated by semicolons. You must use the short form of the macro name for some macros because WinHelp does not support the long name.

HELP_CONTENTS (API) or cdlHelpContents (CD) (3 or &H3) displays the topic specified by the Contents option in the [OPTIONS] section of the .HPJ file. This command is for backward compatibility. New applications should provide a .CNT file and use the **HELP_FINDER** command. *dwData* is ignored. Set it to 0.

HELP_CONTEXT (API) or cdlHelpContext (CD) (1 or 0x001) displays the topic identified by the specified context identifier defined in the [MAP] section of the .HPJ file. *dwData* is an unsigned long integer containing the context identifier for the topic.

HELP_CONTEXTPOPUP (API) or cdlHelpContextPopup (CD) (8 or &H8) displays the topic identified by the specified context identifier defined in the [MAP] section of the .HPJ file in a pop-up window. *dwData* is an unsigned long integer containing the context identifier for a topic.

HELP_FORCEFILE (API) or cdlHelpForceFile (CD) (9 or &H9) ensures that WinHelp is displaying the correct help file. If the incorrect help file is being displayed, WinHelp opens the correct one; otherwise there is no action. *dwData* is ignored. Set it to 0.

HELP_HELPONHELP (API) or cdlHelpHelpOnHelp (CD) (4 or &H4) displays help on how to use Windows Help, if the **WINHLP32.HLP** file is available. *dwData* is ignored. Set it to 0.

HELP_INDEX (API) or cdlHelpIndex (CD) (3 or &H3) is the same as HELP_CONTENTS.

HELP_KEY (API) or cdlHelpKey (CD) (257 or &H101) displays the topic in the keyword table that matches the specified keyword, if there is an exact match. If there is more than one match, displays the Index with the topics listed in the **Topics Found** list box. *dwData* is the a string containing the keyword. Multiple keywords in this string must be separated by semicolons.

HELP_PARTIALKEY (API) or cdlHelpPartialKey (CD) (261 or &H105)displays the topic in the keyword table that matches the specified keyword, if there is an exact match. If there is more than one match, displays the **Topics Found** dialog box. To display the Index without passing a keyword, provide an empty string. *dwData* is the a string containing the keyword. Multiple keywords must be separated by semicolons.

HELP_QUIT (API) or cdlHelpQuit (CD) (2 or &H2) informs WinHelp that it is no longer needed. If no other applications have asked for help, Windows closes WinHelp. *dwData* is ignored. Set it to 0.

HELP_SETCONTENTS (API) cdlHelpSetContents (CD) (5 or &H5) specifies the Contents topic. WinHelp displays this topic when the user clicks the **Contents** button if the help file does not have an associated .cnt file. *dwData* is an unsigned long integer containing the context identifier for the Contents topic.

HELP_SETINDEX (API) cdlHelpSetIndex (CD) (5 or &H5) is the same as HELP_SETCONTENTS.

Additional commands are available through the WinHelp API. See the *Microsoft Help Workshop Help* for descriptions of these additional commands.

WinHelp API Examples

In the examples below:

- "frmMain.hWnd" is the hWnd for the main form in your program.
- Its assumed that you have set the full path and file name for the help file in App.HelpFile.
- "TopicID" is the ID for the desired topic.

To display the Contents topic:

```
WinHelp(frmMain.hWnd, App.HelpFile, HELP_CONTENTS, ByVal 0&)
```

To display the last tab used in the Help Topics dialog box:

```
WinHelp(frmMain.hWnd, App.HelpFile, HELP_FINDER, ByVal 0&)
```

To display the Search dialog:

```
WinHelp(hWnd, App.HelpFile, HELP_PARTIALKEY, "")
```

To display a specific topic in a standard window:

```
WinHelp(hWnd, App.HelpFile, HELP_CONTEXT, ByVal CLng(TopicID))
```

To display a specific topic in a popup:

```
Winhelp(hWnd, App.HelpFile, HELP_CONTEXTPOPUP, \
        ByVal CLng(TopicID))
```

To display a topic or topics using a K Keyword:

```
Winhelp(hWnd, App.HelpFile, HELP_KEY, ByVal keyWord$)
```

To display a topic or topics using an A Keyword after testing to see if the keyword exists:

```
helpCommand$ = "IF(KL(`" & keyWord$ & "', 4, `', `'), \
        `KL(`" & keyWord$ & "', 1, `', `')')"
WinHelp(hWnd, App.HelpFile, HELP_COMMAND, ByVal helpCommand$)
```

To force the help file closed:

```
Winhelp(hWnd, App.HelpFile, HELP_QUIT, ByVal 0&)
```

Using the Common Dialog Control to Call WinHelp

You can use the Common Dialog control to call WinHelp without having to use Windows API calls. If you are not already using the Common Dialog control, just put a copy on the toolbar (it's invisible) or some oter convenient place in you main window.

The relevant properties and methods are:

- **HelpFile** sets the help file to open.
- **HelpCommand** sets the help command to use. There is a Visual Basic constant corresponding to most of the WinHelp API call *uCommand* values. For a list of possible values, see *Help Command Constants* below.
- **HelpContext** sets the topicID if HelpCommand is set to a value that requires a topic ID
- **HelpKey** sets the K Keyword to display topics for. You must set **HelpCommand** = **cdlHelpKey** when you use this property.
- **ShowHelp** sends the help command according to current property settings.

A typical help call to open a topic would look like this:

```
With frmMain.CommonDialog1
   .HelpFile = App.HelpFile
   .HelpContext = topicID
   .HelpCommand = cdlHelpContext
   .HelpKey = ""
   .ShowHelp
End With
```

Calling WinHelp from a Command Line

You can use the Shell command to call WinHelp. Each time you use this command a new instance of WinHelp is opened. The Shell command looks something like:

```
Shell(WinHelpCommandLine, WindowStyle)
```

The syntax for *WinHelpCommandLine* is shown below.

WindowStyle is optional and specifies how to open the window. See Visual Basic documentation for values.

The command line syntax for calling WinHlp32.exe is as follows:

```
winhlp32.exe [[-H] [-G[n]] [-W window] [-K keyword] [-P pop-up]
  [-N contextNum] [-I topicID] helpFile]
```

-H displays the Winhlp32.hlp help file.

-G[n] creates a configuration (.gid) file and quits. If a number is specified, it determines which extensible tab to display by default the first time the help file is opened. A value of 1 would be the first tab beyond the Find tab. This command cannot be used with -S.

-S creates a configuration (.gid) file without showing an animated icon. Cannot be used with -G.

-W *window* specifies the window for displaying the topic. This command cannot be used with -P.

-P specifies that the topic will be shown in a pop-up window. This command cannot be used with -W. You must use the -P switch in combination with the -N (context number) or -I (topic ID) switch, depending on whether you want to specify the context number (from the [MAP] section of the HPJ file) or the topic ID string (from the # footnote of the topic).

-K *keyword* specifies the topic to open using a keyword. This command cannot be used with -I or -N.

-N *contextNum* specifies the topic to open using a topic number, which must be defined in the [MAP] section of the HPJ file. This command cannot be used with -I or -K.

-I *topicID* specifies the topic to open using a topic ID string (# footnote in the topic). This command cannot be used with -N or -K.

helpFile specifies the help file to open. Unless you are calling WinHelp from the directory with the help file, this must be a full path and file name. If you don't specify a help file, the File Open dialog box appears.

Tips:

- If you use long file names with spaces, you must enclose the entire path and file name in quotes.

Training Cards from Visual Basic

You cannot implement training cards from a Visual Basic application in its native form, since it does not support message loops. However, David Liske has created a subclassing module that will let you accomplish this. This modules is advertised as an HTML subclassing module, but since it intercepts the system WM_HELP messages you can use the subclassing portion to intercept WinHelp messages as well as HTML Help messages.

Resources

All of the following plus links to other reference sites about online user assistance are available through the Shadow Mountain Tech Web site (www.smountain.com).

On the Help/Connecting page:

- Latest update of this document and other documents about connecting online help to your program
- *Programmer's Reference to WinHelp* (by Don Lammers and Paul O'Rear)
- Sample code, including API definition modules.
- David Liske's help subclassing tutorial and modules

On the Help/Bugs & FAQ page:

- *WinHelp 4.0 Unofficial Bug and Anomaly List* (currently maintained by Don Lammers)
- *WinHelp FAQ File* (by Charlie Munro)