Connecting HTML Help to C++/MFC Programs

by Don Lammers

Last updated 2001-02-22

Presented with author's permission by

Shadow Mountain Tech

Contents

Scope

Although optimized for WinHelp (as of Visual Studio 6), MFC provides several hooks for easily connecting to any online help, including HTML Help and HTML based help. In addition, you can call the HtmlHelp API from anywhere in your program for additional access to help. The *Connecting Context Sensitive Help to C++/MFC Programs* section shows how to connect to the built in hooks that MFC provides, and is therefore specific to programming with MFC. The remainder of the document should be easily adaptable to any version of C++, as it sets forth the Windows calls and command line interface for HTML Help.

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:

Microsoft HTML 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 HTML Help to C++/MFC Programs

MFC provides for calling WinHelp topics using the WinHelp function of the CWnd class, from which the Main Frame window of your program is derived if you use the App Wizard. In addition you can override dialog box routines to get control level F1 or right click help if needed. In order to use HTML Help you must override the WinHelp function and directly call the HtmlHelp API.

This section shows the steps necessary to hook context sensitive HTML Help into a program created using MFC.

Create a Program with Built-In Help

Since you need to override the automated hooks anyway, and the App Wizard also creates WinHelp files which you will need to remove from your project file, the only reason to have the App Wizard create the hooks is to get the help framework to convert to HTML or HTML Help. If the help author has a tool capable of this conversion (all of the current help authoring tools can do this) it may save some time in setting up the initial file.

To create a new project with context sensitive help support:

1. From the Visual C++ **File** menu, choose **New**.
2. Choose **MFC AppWizard (.exe)**.
3. When given the option, make sure that the **Context Sensitive Help** box is checked.
4. Continue through the process to set up the program the way you want.
5. Remove the .hlp file and folder that the App Wizard created for you. The help author will want these files.

This creates the following:

- An application with a menu and (if you requested it) a toolbar.
- A WinHelp file with topics for each of the automatically generated menu items and toolbar buttons. The help author can use this framework as a starting point for conversion to HTML Help or HTML based help.
- A .hm file with the topic ID to number mapping for the help file.

> Automatically generated .hm files may not be in the correct format for HTML Help. HTML Help requires constant definitions to be in standard C++ #define format. Check these files before giving them to the help author.

- An operational What's This? Help button on the toolbar. Clicking this button puts the program into What's This? Help mode. Clicking on a menu or control then opens its help.
- **Shift-F1** hooked up to put the program into What's this? Help mode.
- **F1** hooked up to open general help for the child window with focus or the control (but not menu) on which the user is currently clicking.
- When you create a dialog box, **F1** hooked up to open a general help topic for the whole dialog box.
- All help topics opened in the main WinHelp window.

Note:

- The above creates support for WinHelp. The following sections show how to override this to call HTML Help or HTML based help.
- Since no dialog boxes are created in this process, you will need to add context sensitive help support for each dialog box as you create it.
- Once you have the initial help file and .hm file, it is usually easier to have the help author add new topics using their preferred help authoring tool. You can still have the program generate the framework help each time you build, because that will give you appropriate additional topic mapping entries.

Set the Help File Name

The default help file name for an MFC generated program is the name of the executable with a .hlp extension. Since you will be overriding the WinHelp function, this variable will never be used by MFC. On the other hand, it's a handy place to store the help file name—especially if you only have a single help file.

To set the help file name for MFC calls:

- Set the **m_pszHelpFilePath** variable member of the application class in the **InitInstance** function. **m_pszHelpFilePath** is a public variable of type **const char***. According to MS, you should first free the memory associated with the default string (the app name with a .hlp extension).
  ```
  //First free the string allocated by MFC at CWinApp startup.
  //The string is allocated before InitInstance is called.
  free((void*)m_pszHelpFilePath);
  //Change the name of the help file.
  //The CWinApp destructor will free the memory.
  m_pszHelpFilePath=_tcsdup(_T("d:\\somedir\\myhelp.chm"));
  ```

Tips:

- **m_pszHelpFilePath** 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 to call from the program you should put the code for changing help file names 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 HTML Help Can Find the Help File

Make sure that HTML Help 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 (or both).

> If the help system contains more than one .chm file and a master Table of Contents (.hhc file), and not all of these are in the same folder, it will be necessary to register the help file and the master TOC to make the master TOC work, regardless of any other issues.

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

- Register the help file by creating a string value under HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\HTMLHelp. 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 **m_pszHelpFilePath**, even if the help file is in the same folder as the program executable.

Manually Add the Context Hooks

If you already have a program and need to add help, you can manually edit the files to create the necessary hooks.

To add context sensitive help manually:

1. Add the necessary message map entries in the message map of the Main Frame class. If you created the program with the MFC App Wizard and specified that you wanted context help, these items should already be in the file.
   ```
   ON_COMMAND(ID_HELP_FINDER, CMDIFrameWnd::OnHelpFinder)
   ON_COMMAND(ID_HELP, CMDIFrameWnd::OnHelp)
   ON_COMMAND(ID_CONTEXT_HELP, CMDIFrameWnd::OnContextHelp)
   ON_COMMAND(ID_DEFAULT_HELP, CMDIFrameWnd::OnHelpFinder)
   ```
2. Copy the What's This? Help icon into a button on your toolbar. This icon is available as a bitmap in C:\Program Files\Microsoft Visual Studio\Common\Graphics\Bitmaps\OffCtlBr\Small\Color\help.bmp.
3. Assign ID_CONTEXT_HELP to the What's This? Help button.

Note:

- This creates the same basic hooks in the program as the App Wizard, but does not generate a framework help file or the .hm file with the topic ID mapping. You will need to create a list of IDs for the help author to use.

Override the Default Context Help Behavior

The default WinHelp call made from the hooks automatically created by the App Wizard opens each help topic in the main WinHelp window. However, you want this program to call HTML Help or HTML based help. In addition, the current recommended behavior for this help is:

- Control level help is opened in a popup.
- Window level help is opened in a normal window (usually the main tri-pane window).

To create these conditions, you must override the default functions.

To create currently recommended behavior:

1. In the Main Frame class, create an **OnHelpInfo** function using the Class Wizard:
   ```
   BOOL CMainFrame::OnHelpInfo(HELPINFO* pHelpInfo)
   {
     return TRUE;
   }
   ```
   This effectively disables the function, and is necessary because, in combination with the next override, all topics accessed through the **F1** key will be called twice. Unfortunately you cannot use the **OnHelpInfo** function in the Main Frame since (unlike in dialog boxes) it is triggered only by the **F1** key and not by **Shift-F1** or the key.

2. In the Main Frame class, create a **WinHelp** function (yes, WinHelp, not HtmlHelp) using the Class Wizard:

```
void CMainFrame::WinHelp(DWORD dwData, UINT nCmd)
{
  CWinApp* theApp = AfxGetApp();
  CString helpFilePath = theApp->m_pszHelpFilePath;
  switch ( dwData ) {
  case HIDR_MAINFRAME:
  case HIDR_MFCWIZTYPE:
  case HIDD_ABOUTBOX:
  case HIDD_TESTDLG:
    //Topics that need to go into the main window.
    HtmlHelp(m_hWnd, helpFilePath, HH_HELP_CONTEXT, dwData);
    break;
  case WHATEVER
    //Topics that need to go into a different secondary window
    ...
    break;
  default:
    //All the rest are popups
    HtmlHelp(m_hWnd, helpFilePath, HH_HELP_CONTEXT, dwData);
    break; }
}
```

Tips:

- It is often easier to create a separate function or class to process the help calls and open topics in the correct window. This can be part of the main program or can be created as a separate DLL. The main advantage of a DLL is that it can be called from other programs if necessary.
- If you want the help author to be able to control the help file and window called by this function, class, or DLL, put the help file and window definitions in an INI file (you can use the topic number as the key) and look them up before calling Help. If you include the help type as an INI file entry you can easily call multiple types of user assistance from the same program.

Activate Control Level Help for a Dialog Box

You can add control level help (What's This? Help) to a dialog box by overriding the **OnHelpInfo** function for the dialog box class.

To add What's This? Help to a dialog box:

1. In the dialog box class, create a **OnHelpInfo** function using the Class Wizard:

```
BOOL cTestDlg::OnHelpInfo(HELPINFO* pHelpInfo) {
  //Get the control ID
  DWORD cControlID = pHelpInfo->iCtrlId;
  //Only proceed if it's not a static control
  if (cControlID != 65535)  //65535 is IDC_STATIC
  {
    //Get the context ID for the control
    DWORD cTopic = pHelpInfo->dwContextId;
    //Get the hWnd for the app
    HWND hwndApp = m_hWnd;
    //Create an app object for this app
    CWinApp* theApp = AfxGetApp();
    //Get the help file name
    CString helpFilePath = theApp->m_pszHelpFilePath;
    //Call the HtmlHelp API. This bypasses all MFC code.
    HtmlHelp(m_hWnd, helpFilePath, HH_DISPLAY_TEXT_POPUP, cTopic);
  }
  return TRUE;
}
```

2. In the **Extended Styles** tab of its property sheet, make sure that **Context help** is on.

Note:

- You cannot add the What's This? Help button to any resizeable window or to a tool window.

Tip:

- If you want the What's This? Help button in a tool window, make it a Fixed Dialog window instead. The only difference I've ever seen is that the title bar is slightly smaller on a tool window.

The HELPINFO Structure

Information about the control that is calling help is contained in the HELPINFO structure passed to the **OnHelpInfo** function.

The HELPINFO structure is defined as follows:

```
typedef struct tagHELPINFO {
  UINT     cbSize;
  int      iContextType;
  int      iCtrlId;
  HANDLE   hItemHandle;
  DWORD    dwContextId;
  POINT    MousePos;
}  HELPINFO, FAR *LPHELPINFO;
```

cbSize is the structure size, in bytes.

iContextType is the type of context for which Help is requested. This can be either HELPINFO_MENUITEM (help was requested from a menu item) or HELPINFO_WINDOW (help was requested from a control or window).

iCtrlId is the identifier of the window or control if *iContextType* is HELPINFO_WINDOW, or the menu identifier if *iContextType* is HELPINFO_MENUITEM.

hItemHandle is the identifier of the child window or control if *iContextType* is HELPINFO_WINDOW, or identifier of the associated menu if *iContextType* is HELPINFO_MENUITEM.

dwContextId is the help context identifier of the window or control.

MousePos is a POINT structure containing the screen coordinates of the mouse cursor.

Create a Single Help Calling Function

If you need to handle anything that is not standard MFC functionality it is usually easier to create a single help calling function (I usually call it OpenTopic) somewhere in the program or in a separate DLL. In each of the dialog box **OnHelpInfo** functions, in the Main Frame **WinHelp** function, and anywhere that you would have called the HtmlHelp API directly, you call this function instead. This puts all of the logic behind determining how to open a topic in one place and makes it easy to change as needed. You can easily treat other elements of the online user assistance (multimedia, Acrobat files, etc.) as topics by assigning them a topic number and handling the actual calls to them in this function. And of course you can mix help types (for instance HTML Help and WinHelp) if this is necessary for some reason.

Most of the time this function needs only two parameters: the main window hWnd and the topic number. However, if you have multiple programs potentially calling into the same help system, you may want to include a program ID as a third parameter so you know who generated the request.

A sample DLL version of this topic is shown below

GetFullFileSpec is a function that extracts the path from the second parameter (in this case the DLL name) and appends the first parameter (in this case the help file name and window) to create the necessary file name and window combination to call HTML Help.

ChangeExtension is a function that replaces the extension on the first parameter with the string in the second parameter.

To initialize and get the DLL name and location:

```
int __declspec(dllexport) WINAPI OpenTopic (HWND hwndApp, DWORD Topic)
{
  //Get the instance handle for the DLL
  CWinApp* theApp = AfxGetApp();
  HINSTANCE dllInstance = theApp->m_hInstance;
  //Get the name of the DLL
  char cDLLName[512];
  GetModuleFileName(dllInstance, cDLLName, 512);
  char cHelpCall[512];
```

To intercept the call and display the topic number for troubleshooting:

This simply shows the topic number called. To activate it you need to put a ShowTopicNumber=1 entry in the [Genera] section of a text file that has the DLL path and name but with a .dat extension. This little bit of code can often save hours of arguing over where the context help calls are broken, since it shows the exact numeric value being sent by the program to help.

```
  //See if we are supposed to display the topic number for debugging
  char iniFile[256];
  ChangeExtension(cDLLName, ".dat", iniFile);
  if (GetPrivateProfileInt("General", "showTopicNumber", 0, \
        iniFile) != 0) {
    char buffer[256];
    wsprintf(buffer, "About to open topic number %d \
        (%X hex)", Topic, Topic);
    MessageBox( NULL, buffer, \
            "Topic Number", MB_OK | MB_ICONINFORMATION );
  }
```

To open the Help Topics dialog box:

See *Calling the Help Finder* below for more detail.

```
if (Topic == 999)
{  //Open the Help Topics dialog box
  GetFullFileSpec("helpfile.hlp", cDLLName, cHelpCall);
  HtmlHelp(hwndApp, cHelpCall, HH_DISPLAY_TOC, NULL);
  return 0;
}
```

To open a warning dialog box called as though it is a topic:

This section opens a warning message dialog box with multiple possible returns. The dialog box is modal (which is one reason why we did not use a help topic to warn the user) and since the DLL is called from the application the user must respond before the program can continue. When the program determines that a warning condition exists, it calls OpenTopic with the appropriate topic number.

In the original version of this warning dialog, all text for the dialog box was stored in a text file so it could be easily edited by the help author.

m_WarningText is the key that the dialog box will use to find the topic text in a text file.

```
else if (Topic > 0 && Topic < 50)
{ //Warnings (warning agent popup topics)
  char cTopic[64];
  wsprintf(cTopic, "topic%lu", Topic);
  int noShow = GetPrivateProfileInt(cTopic,"NoShow", 0, iniFile);
  if (noShow == 0) {
    CWarnDlg cWarningDialog;
    cWarningDialog.m_WarningText = _T(cTopic);
    ......
    //Open the dialog box
    int branch = cWarningDialog.DoModal();
    return branch; }
  else {
    return 1; }
}
```

To ignore certain topic numbers:

```
else if (Topic == 65535)
{  // Generic Control ID--ignore
  return 0; }
else if (Topic > 140000)
{ //Ignore these topic requests.
  //262144 = Separator between info bar and status bar
  //262150 = Horizontal scroll bar
  ......
  return 0; }
```

To open a topic in the tri-pane window:

```
     else if (Topic > 130000)
     { //Secondary window Help for dialog box overviews.
       HtmlHelp(hwndApp, cHelpCall, HH_HELP_CONTEXT, (DWORD)Topic);
       return 0; }
```

To open a topic in a secondary window:

```
     else if (Topic > 140000)
     { //Secondary window Help for dialog box overviews.
       GetFullFileSpec ("helpfile.chm>second", cDLLName, cHelpCall);
       HtmlHelp(hwndApp, cHelpCall, HH_HELP_CONTEXT, (DWORD)Topic);
       return 0; }
```

To open a topic as a popup:

```
     else
     {  // Popup help for menu items and controls.
       GetFullFileSpec ("helpfile.hlp", cDLLName, cHelpCall);
       HtmlHelp(hwndApp, cHelpCall, HH_DISPLAY_TEXT_POPUP, (DWORD)Topic);
       return 0; }
     return 0;
   }
```

Notes:

- The above code filters out any messages coming from controls assigned an ID of IDC_STATIC. If you want static controls (labels) to also have context help, they must each be assigned unique identifiers.
- Since all controls in a program should have unique IDs, you may be able to use the control ID rather than the context ID for this function. This is a smaller number and easier to deal with in the help mapping (sorry, just being lazy).

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 popup 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. When the user right clicks a control, and before opening a popup menu for the control, save the topic number to call.
2. Implement whatever code is needed to open the popup menu with the appropriate items listed.
3. In the Click event for the **What's This?** menu item, call the HtmlHelp 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 popup menu with a single What's This? item looks a bit strange, and this saves the user one click.

Opening HTML Help Directly from Your C++ Code

You can programmatically open 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.

HtmlHelp API Syntax

The **HtmlHelp** function starts HTML Help and passes additional data indicating the nature of the help requested by the application. To use the following API calls you must attach htmlhelp.h and include htmlhelp.lib in you links. These files are available in subfolders when you install the *Microsoft HTML Help Workshop*.

```
BOOL HtmlHelp( 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 **HtmlHelp** 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 HTML Help is to display. For some commands it can also contain the file name of the specific topic. The format for this string is:

```
Helpfile.chm[::/Topic.htm][>Window name]
```

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 (.hhp) file.

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

dwData specifies additional data. The value used depends on the value of the *uCommand* parameter. For a partial list of possible values, see *Help Command Constants* below. For a full list, see the *Microsoft HTML Help Workshop Help*.

HtmlHelp Command Constants

The following list shows the possible values for the *uCommand* parameter, its value, its action, and the corresponding formats of the *dwDat*a command. For details about each of the structures referred to in this section, see *Help Command Structures*.

HH_DISPLAY_TOPIC (0 or 0x0000) selects the **Contents** tab in the navigation pane of the HTML Help viewer (tri-pane window) and optionally displays a specified topic. If the topic is not specified in *lpszHelp* then *dwData* is a string containing the file name of the topic to open (including relative folder information if necessary). Otherwise *dwData* is 0& (NULL). If *dwData* is 0, you can specify the topic and/or defined window in *lpszHelp* using the following format:

```
    Helpfile.chm[::/Topic.htm][>Window name]
```

HH_DISPLAY_TOC (1 or 0x0001) is the same as HH_DISPLAY_TOPIC.

HH_DISPLAY_INDEX (2 or 0x0002) selects the **Index** tab in the navigation pane of the HTML Help viewer (tri-pane window) and searches for a keyword. *dwData* is a string containing keyword to search for.

HH_DISPLAY_SEARCH (3 or 0x0003) selects the **Search** tab in the navigation pane of the HTML Help viewer (tri-pane window) and performs a search. *dwData* an HH_FTS_QUERY structure that specifies the search parameters.

HH_KEYWORD_LOOKUP (13 or 0x000D) searches for one or more keywords in a compiled HTML Help (.chm) file. *dwData* is an HH_AKLINK structure containing the keywords to search for and the actions to be taken if no matches are found.

HH_DISPLAY_TEXT_POPUP (14 or 0x000E) opens a pop-up window and displays one of the following:

- An explicit text string.
- A text string based on a resource ID
- A text string based on a text file contained in a compiled HTML Help (.chm) file.

To use an explicit text string or a string based on a resource identifier, set *lpszHelp* to an empty string. To use a text string from a file contained in a compiled HTML Help (.chm) file, set *lpszHelp* to the name of the .chm file and the text file within the .chm file. In either case *dwData* is an HH_POPUP structure.

HH_HELP_CONTEXT (15 or 0x000F) 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.

HH_CLOSE_ALL (18 or 0x0012) closes all windows that have been opened directly or indirectly by the calling program. *hWndMain* must be 0 (zero).*lpszHelp* must be an empty string. *uCommand* must be 0 (zero).

HH_ALINK_LOOKUP (19 or 0x0013) searches for one or more Associative link (Alink) names in a compiled HTML Help (.chm) file. *dwData* is an HH_AKLINK structure containing the names to search for and the actions to be taken if no matches are found.

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

Help Structures

The following list shows the data that goes into each of the structures referred to in the *Help Command Constants* section.

The HH_POPUP structure is defined as follows:

```
typedef struct tagHH_POPUP {
  int cbStruct;
  HINSTANCE hinst;
  UINT idString;
  LPCTSTR pszText;
  POINT pt;
  COLORREF clrForeground;
  COLORREF clrBackground;
  RECT rcMargins;
  LPCTSTR pszFont;
} HH_POPUP;
```

cbStruct is an Long variable which specifies the size of the structure. You must always fill in this value before passing the structure.

hinst is a Long variable containing the instance handle for the string resource.

idString is a Long variable specifying the string resource ID, or the text ID if *pszFile* is specified in the HtmlHelp call.

pszText is the explicit string to display. To display this string you must set *idString* to 0 (zero).

pt is a POINTAPI structure that specifies the top center of the popup window.

clrForeground specifies the foreground (text) color used in the pop-up. Use a VB color constant or a number in the format &HBBGGRR.

clrBackground specifies the background color used in the pop-up. Use a VB color constant or a number in the format &HBBGGRR.

rcMargins is a RECT structure indicating the amount of space between edges of window and text. Use -1 for each member to ignore.

pszFont is a string containing the font to use in the pop-up. The string is in the following format:

```
facename[, point size[, char set[, BOLD ITALIC UNDERLINE]]]
```

You can skip an attribute by entering a comma for the attribute.

Note:

- This structure contains standard Windows POINTAPI and RECT structures, which must be defined in the project. You can get these structures from the API Text Viewer that came with Visual Basic.

The HH_AKLINK structure is defined as follows:

```
typedef struct tagHH_AKLINK {
  int cbStruct;
  BOOL fReserved;
  LPCTSTR pszKeywords;
  LPCTSTR pszUrl;
  LPCTSTR pszMsgText;
  LPCTSTR pszMsgTitle;
  LPCTSTR pszWindow;
  BOOL fIndexOnFail;
} HH_AKLINK;
```

cbStruct is an Long variable which specifies the size of the structure. You must always fill in this value before passing the structure.

fReserved is reserved. Must be False.

pszKeywords is a string containing the keywords to search for. Separate multiple keywords with simicolons.

pszURL is a string containing the name of the topic file to show if no matches are found. May be NULL. Use the following format for this string:

```
Helpfile.chm[::/Topic.htm][>Window name]
```

pszMsgText is a string containing the message box text to display if *fIndexOnFail* is False and *pszURL* is an empty string.

pszMsgTitle is a string containing the caption of the message box which will be displayed if *fIndexOnFail* is False and *pszURL* is an empty string.

pszWindow is a string containing the name of the secondary help window in which to display one of the following:

- The selected topic if the lookup yields one or more matching topics.
- The topic specified in *pszURL* if the lookup fails and a topic is specified in *pszURL*.
- The **Index** tab if the lookup fails and *fIndexOnFail* is True.

fIndexOnFail is True if you want to display the keyword in the Index tab of the HTML Help viewer (tri-pane window) if the lookup fails

The HH_FTS_QUERY structure is defined as follows:

```
typedef struct tagHH_FTS_QUERY
{
  int      cbStruct;
  BOOL     fUniCodeStrings;
  LPCTSTR  pszSearchQuery;
  LONG     iProximity;
  BOOL     fStemmedSearch;
  BOOL     fTitleOnly;
  BOOL     fExecute;
  LPCTSTR  pszWindow;
} HH_FTS_QUERY;
```

cbStruct is an Long variable which specifies the size of the structure. You must always fill in this value before passing the structure.

fUniCodeStrings is a Long variable which is non-zero if all strings are Unicode.

pszSearchQuery is a string containing the search query.

iProximity is a Long variable which specifies word proximity.

fStemmedSearch is a Long variable which is non-zero if you want a stemmed search.

fTitleOnly is a long variable which is non-zero if you want to search titles only.

fExecute is a long variable which is non-zero to initiate the search.

pszWindow is a string which specifies the window in which to display the results.

HtmlHelp API Examples

In the examples below:

- "m_hWnd" is the window handle for the main form in your program.
- Its assumed that you have set the full path and file name for the help file in **m_pszHelpFilePath**.
- "TopicID" is the ID for the desired topic.

To display the Contents tab in the navigation pane and open the default topic:

```
HtmlHelp(m_hWnd, m_pszHelpFilePath, HH_DISPLAY_TOC, ByVal 0&);
```

To display the Contents tab in the navigation pane and open a specific topic:

```
HtmlHelp(m_hWnd, m_pszHelpFilePath & \
        "::/RelativeFolder\\TopicFileName.htm", \
        HH_DISPLAY_TOPIC, ByVal 0&);
  -OR-
HtmlHelp(m_hWnd, m_pszHelpFilePath, HH_DISPLAY_TOPIC, \
        ByVal "RelativeFolder\\TopicFileName.htm");
```

To display the Index tab in the navigation pane:

```
HtmlHelp(hWnd, m_pszHelpFilePath, HH_DISPLAY_INDEX, "");
```

To display the Search tab in the navigation pane:

```
HH_FTS_QUERY searchIt;
searchIt.cbStruct = len(searchIt);
searchIt.fUniCodeStrings = 0&;
searchIt.pszSearchQuery = "";
searchIt.iProximity = 0&;
searchIt.fStemmedSearch = 0&;
searchIt.fTitleOnly = 0&;
searchIt.fExecute = 0&;
searchIt.pszWindow = "";
HtmlHelp(m_hWnd, m_pszHelpFilePath, HH_DISPLAY_SEARCH, searchIt);
```

To display a specific topic in a standard window:

```
HtmlHelp(m_hWnd, m_pszHelpFilePath, HH_HELP_CONTEXT, \
        ByVal CLng(TopicID));
```

To display a specific topic in a popup:

```
HtmlHelp(m_hWnd, m_pszHelpFilePath, HH_HELP_CONTEXTPOPUP, \
        ByVal CLng(TopicID));
```

To display a topic or topics using a K Keyword:

```
HH_AKLINK keyData;
KeyData.cbStruct = len(keyData);
KeyData.fReserved = 0&;
KeyData.pszKeywords = "cats";
KeyData.pszUrl = "";
KeyData.pszMsgText = "No such keyword found";
KeyData.pszMsgTitle = "Keyword Not Found";
KeyData.pszWindow = "";
KeyData.fIndexOnFail = 0&;
HtmlHelp(m_hWnd, m_pszHelpFilePath, HH_KEYWORD_LOOKUP, keyData);
```

To display a topic or topics using an A Keyword:

```
HH_AKLINK keyData;
KeyData.cbStruct = len(keyData);
KeyData.fReserved = 0&;
KeyData.pszKeywords = "cats";
KeyData.pszUrl = "";
KeyData.pszMsgText = "No such associative link found";
KeyData.pszMsgTitle = "Link Not Found";
KeyData.pszWindow = "";
KeyData.fIndexOnFail = 0&;
HtmlHelp(m_hWnd, m_pszHelpFilePath, HH_ALINK_LOOKUP, keyData);
```

To force all HTML Help windows closed that were opened by this app:

```
HtmlHelp(0&, "", HH_CLOSE_ALL, ByVal 0&);
```

Training Cards from C++

Training cards allow the help system to communicate directly with the program. This section describes the general procedures for setting up training cards using C++.

Sending Training Card Messages from Help

The help author inserts the HTML Help ActiveX control into training card topics to send information to programs via the WM_TCARD message. The program must intercept the WM_TCARD message and determine the required action.

To insert the HTML Help ActiveX control into a topic:

1. Open the HTML topic file in the HTML Help Workshop.
2. Place the cursor where you want the functionality to appear.
3. From the **Tags** menu, choose **HTML Help Control...** and follow the instructions to insert the control.

You can use the TCard command of the ActiveX control to send a WM_TCARD message from the HTML Help file. This command works only from within a compiles HTML Help (.chm) file. the following properties are relevant to this command:

The **Button** property sets the button type.

The **Command** property needs to be set to "TCard."

The **Font** property sets the font for the control.

The **Item1** property sets the *wPARAM* and the *lPARAM* parameter values for the **WM_TCARD** message. *wPARAM* usually identifies a user action and must be numeric. *lPARAM* may contain additional data, depending on the value of *wPARAM*, and may be numeric or string. See the WM_TCARD message section below for more details.

The **Text** property sets the text to display as the link or button.

The WM_TCARD Message

The program must intercept the WM_TCARD message from the help file and determine the correct action to take.

The *wParam* value for the WM_TCARD message is one of the following:

IDABORT	The user clicked an authorable **Abort** button.
IDCANCEL	The user clicked an authorable **Cancel** button.
IDCLOSE	The user closed the training card.
IDHELP	The user clicked an authorable **Help** button.
IDIGNORE	The user clicked an authorable **Ignore** button.
IDOK	The user clicked an authorable **OK** button.
IDNO	The user clicked an authorable **No** button.
IDRETRY	The user clicked an authorable **Retry** button.
IDYES	The user clicked an authorable **Yes** button.
HELP_TCARD_DATA	The user clicked an authorable button. The *lParam* parameter contains a long integer specified by the help author.
HELP_TCARD_NEXT	The user clicked an authorable Next button.
HELP_TCARD_OTHER_CALLER	Another program has requested training cards.

If *wParam* is **HELP_TCARD_DATA**, *lParam* is the number indicated by the TCARD macro. Otherwise *lParam* is 0 (zero).

Processing TCard Commands in the Program

The basic procedure for setting up TCard processing follows. Please see the Windows SDK documentation for details on how to use the **SetWindowsHookEx** and **UnhookWindowsHookEx** functions, and on the required hook processing within your hook procedure.

To intercept and process the WM_TCARD message:

1. With the help author, decide on the codes that will be used for any function that is not predefined.
2. From the program, open HtmlHelp using the WinHelp command. Call the HtmlHelp API directly rather than going through MFC to ensure that MFC does not intercept and process the command somewhere.
   ```
   HtmlHelp(m_hWnd, m_pszHelpFilePath, HELP_CONTEXT, TopicID);
   ```
3. Call the **SetWindowsHookEx** function with a **WH_CALLWNDPROC** hook type to set up a hook procedure to monitor WM_TCARD messages sent from help.
4. When the hook procedure detects the WM_TCARD message, process the message to determine the necessary actions.
5. Make sure you release the hook procedure using **UnhookWindowsHookEx** before closing the program.

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)