A Help Think Press Publication

Delphi Programming for the HTML Help API

by Robert Chandler
The Helpware Group
http://www.helpware.net

Help Think Press
A Division of Work Write, Inc.
128 South Eastview Avenue
Feasterville, PA 19053 USA
Tel: +1 215.357.3453
Web: http://www.workwrite.com/helpthink/welcome.htm
Email: helpthink@workwrite.com

The publisher welcomes your feedback

This document is published by Help Think Press, a division of Work Write, Inc. Work Write, Inc. Work Write, Inc. specializes in the development of hypertext user assistance as well as training on authoring tools and help design. (For more information, see "About the pubisher" at the end of this document.) Help Think Press welcomes (truly! no kidding!) your comments on these materials and, if we find them to be valid, will make every effort to incorporate them into the next edition.

You can reach Cheryl Lockett Zubak at Work Write, Inc., 128 South Eastview Avenue, Feasterville, PA 19053 USA. Phone: +1 (215) 357-3453; fax +1 (215) 357-0695; email cheri@workwrite.com; website http://www.workwrite.com.

Table of contents

A Help Think Press Publication

Delphi Programming for the HTML Help API

by Robert Chandler
The Helpware Group
http://www.helpware.net

This chapter covers HTML Help programming using Borland's 32-bit Delphi (i.e., Delphi 2 and above). While Delphi does not support HTML Help directly, Delphi is a very rich and flexible programming environment. We will see that it takes very little effort to convert our WinHelp Delphi applications to HTML Help.

The chapter assumes that you have a basic working knowledge of Delphi programming and have glanced at the HTML Help API reference in the Microsoft HTML Help Workshop online help. In this chapter, we will cover such subjects as:

- Connecting to the API
- Basic help commands
- Moving from WinHelp to HTML Help
- Understanding HTML Help programming issues and API commands
- Reviewing the free Delphi code
- Connecting HTML Help systems with your Delphi program

This document and accompanying sample files were developed under contract to:

Help Think Press
A Division of Work Write, Inc.
http://www.workwrite.com/helpthink/welcome.htm

Examples and source code

If you haven't yet installed the HTML Help Workshop, you should do so, since it includes detailed documentation on programming the HTML Help API. You can download it from:

http://msdn.microsoft.com/library/en-us/htmlhelp/html/vsconHH1Start.asp

The example applications, including Delphi source, that are discussed in this document are on the Helpware web site:

http://www.helpware.net/delphi/

Joining the Helpware mailing list is the easiest way to keep up with all the latest updates and news.

As we work through the chapter I will point you towards the appropriate code examples.

Location	Description
\Example 1	Accessing the HH API using static loading. Common HH API calls.
\Example 2	Accessing the HH API using dynamic loading. Common HH API calls.
\Example 3	Hooking help events in Delphi. Moving from WinHelp to HTML Help.
\Example 4	Advanced HTML Help API calls. What's this help.
\Example 5	Embedding Internet Explorer WebBrowser control into Delphi applications.
\Example 6	Embedding HH windows into Delphi applications.
\Example 7	Creating an out-of-process automation server to extend HTML Help.
\Example 8	Finding open HTML Help windows and basic manipulation of HH windows using windows API functions.
\Example 9	Advanced HH window manipulation. Warning: Some of these back door methods of manipulating the HTML Help window may not work with future versions of HTML Help. Use at your own risk. This chapter will not cover this area.
\Example 10	A Delphi port of Ralph Walden's code that reads the file structure of a CHM help file. This chapter will not cover this area..

Location	Description
\Example 11	Calling KeyHelp.ocx Interfaces from Delphi. KeyHelp can be downloaded for free from http://www.keyworks.net.
\Example12	Delphi 6 broke the OnHelp event used to trap all Delphi help events. This example shows you how to fix this bug in Delphi 6.

All examples can be compiled under Delphi 3 or greater, except for: Example 5 contains Delph 3 and Delphi 4 and greater examples; Example 7 requires Delphi 4 or greater; Example 12 is designed for Delph 6 only. Most examples will compile under Delphi 2 if given a little push.

Getting started

While HTML Help requires that you install a minimum of Internet Explorer 3, Internet Explorer 5 or higher is highly recommended. In fact, some HTML Help features, such as word search highlighting, only work if you have Internet Explorer 4 or above. Versions of HTML Help earlier than 1.2 are not very compatible with Delphi due to window parenting problems (more about this later). I also advise Delphi 3 users to upgrade to Delphi 4 or 5 to resolve other problems.

Installing the latest version of both Internet Explorer and HTML Help will ensure that you have the latest bug fixes and optimizations. You will find the latest versions of Internet Explorer, HTML Help, and HTML Help Workshop programs on the Microsoft web site.

The following table will help you to check if your Windows operating system requires an update.

Operating Systems	Internet Explorer version	HTML Help version	Update required?
Windows ME **	5.5	1.32	No
Windows 2000 SP1 **	5	1.31	No
Windows 2000 **	5	1.3	No
Windows 98 SE (2nd Edition)	5	1.21	No
Windows 98	4	1.1a	Yes — HH
Windows 95 OSR2	3	—	Yes — IE and HH
Windows 95	—	—	Yes — IE and HH
Windows NT4/SP3	—	—	Yes — IE and HH

** For these operating systems, you can only update HTML Help runtime files via an official Microsoft Service Pack.

A few notes

- Windows 3.X and Windows NT 3 are not suitable for displaying HTML Help.
- Check the version of Internet Explorer by looking in the Help › About Box. Check the version of HTML Help by opening the About box in the HTML Help Workshop. Alternatively open any CHM file, click the system menu gadget in the top left corner of the window, and select version.
- For applications, the Internet Explorer version number can be read from shdocvw.dll, while the HTML Help version number can be read from hhctrl.ocx. Both these files are installed into the windows system folder. Be aware that hhctrl.ocx is an ActiveX control, and as such can be installed to the windows ActiveX cache when a web page that uses the control is opened. I'll show you how to find hhctrl.ocx later.

Got Windows all setup? Then let's begin.

Connecting to the API

Programmers have access to a range of help commands via a single API function call HtmlHelp(). HtmlHelp() works similarly to the old WinHelp() function. Both take the same type of parameters. The commands, however, are quite different.

The HtmlHelp() function lives in a Dynamic Link Library called hhctrl.ocx and comes in the usual ANSI and UNICODE flavors. You will notice that this library has an OCX file extension. It has some ActiveX functionality, used for embedded HTML Help functionality into the WebBrowser control. Hhctrl.ocx cannot be embedded into other applications such as Delphi, VB or C++. For our purposes, think of hhctrl.ocx as a normal DLL.

Delphi does not ship with an HTML Help import unit, so the first thing we must do is port the basic elements of the C++ header file, htmlhelp.h, to a Delphi unit file. If you have installed the HTML Help Workshop, you can find a copy of htmlhelp.h installed in the include folder. Here is the default location of htmlhelp.h:

```
C:\Program Files\HTML Help Workshop\Include
```

Note: I have already ported htmlhelp.h to Delphi for you. The import unit has the name hh.pas, since a Delphi unit and function name cannot be the same. Hh.pas can be found with the other examples code.

Static versus dynamic loading

You can the access the HtmlHelp() API function in two ways: Static loading by declaring an external function, and dynamic loading using the Windows API functions LoadLibrary() and FreeLibrary(). Neither method requires the DLL to be present at compile time.

Using the static loading method poses two potential problems:

1. Static loading requires that the DLL be present at run-time. Since Windows 95 does not come with HTML Help, this will frustrate many users. Try it yourself. Copy example 1 to a PC that does not have HTML Help installed. The application will not run. If you install Internet Explorer and HTML Help during program installation, this will not be a problem for you.

2. A web page referencing htmlhelp.ocx could download hhctrl.ocx to the Windows ActiveX cache. Now we have a situation where the static load code cannot locate the library even though it is installed.

Let's look at both methods in detail and you can decide which you want to use.

Static loading

Static loading, also known as *implicit dynamic linking*, is performed at application load time by the operating system. It is the simplest and the most common method of importing API functions under Delphi.

Here is the Delphi declaration for accessing the HTML Help API. The three most common commands are also declared. Look inside any import unit such as windows.pas and you will see this kind of code.

```
interface
uses Windows;
  function HtmlHelpA(hwndCaller: HWND; pszFile: PAnsiChar;
    uCommand: UINT; dwData: DWORD): HWND; stdcall;
  function HtmlHelpW(hwndCaller: HWND; pszFile: PWideChar;
    uCommand: UINT; dwData: DWORD): HWND; stdcall;
  function HtmlHelp(hwndCaller: HWND; pszFile: PChar;
    uCommand: UINT; dwData: DWORD): HWND; stdcall;
const
  hhctrlLib = 'hhctrl.ocx';
const
  { Commands to pass to HtmlHelp() }
  HH_DISPLAY_TOPIC = $0000;
  HH_HELP_CONTEXT  = $000F;
  HH_CLOSE_ALL     = $0012;
implementation

  function HtmlHelpA; external hhctrlLib Name 'HtmlHelpA';
  function HtmlHelpW; external hhctrlLib Name 'HtmlHelpW';
  function HtmlHelp; external hhctrlLib Name 'HtmlHelpA';
```

Three function names are declared: *HtmlHelpA* for the ansi-character version of the function, *HtmlHelpW* for the wide-character (unicode) version, and *HtmlHelp*. *HtmlHelp* currently uses the ansichar-version *HtmlHelpA* and is the function we will use in all examples.

Check out Example 1 for the full working version.

Dynamic loading

Dynamic loading, also known as *explicit dynamic linking*, is performed at run-time by making calls to the operating system to load the DLL using the LoadLibrary() and FreeLibrary() functions.

This approach is more complex but will produce a more robust application. Let's walk through the code (below).

It's a bigger slab of code, but there is not much to it, really. It includes a global variable called HHCtrlHandle. The variable is non-zero if the HTML Help library was found and loaded successfully. The code also uses the HTML Help function HtmlHelp(), which is valid only when HHCtrlHandle is non-zero. The code also has two procedures, LoadHtmlHelp() and UnloadHTMLHelp(), which can be called on module initialization and finalization. The first function calls LoadLibrary(), while the other calls FreeLibrary().

LoadLibrary() requires the full path to hhctrl.ocx. Ralph Walden, one of the original authors of HTML Help, suggests that we can find the full path by reading the default value of the following Windows registry key.

```
[HKEY_CLASSES_ROOT\CLSID\{adb880a6-d8ff-11cf-9377-
00aa003b7a11}\InprocServer32]
```

Check out Example 2 for the full working version.

```
interface


uses Windows, Registry;

var
  { =0 if HTML Help could not be loaded}
  HHCtrlHandle: THandle = 0;
{ Externals from HHCTRL.OCX }

var  //functions are invalid if HHCtrlHandle = 0
  HtmlHelpA: function(hwndCaller: HWND; pszFile: PAnsiChar;
    uCommand: UInt; dwData: DWORD): HWND; stdcall;
  HtmlHelpW: function(hwndCaller: HWND; pszFile: PWideChar;
    uCommand: UInt; dwData: DWORD): HWND; stdcall;
  HtmlHelp: function(hwndCaller: HWND; pszFile: PChar;
    uCommand: UInt; dwData: DWORD): HWND; stdcall;
```

```
const
  hhctrlLib = ‘hhctrl.ocx’;
const
  { Commands to pass to HtmlHelp() }
  HH_DISPLAY_TOPIC = $0000;
  HH_HELP_CONTEXT  = $000F;
  HH_CLOSE_ALL     = $0012;

implementation
const hhPathRegKey =
  ‘CLSID\{adb880a6-d8ff-11cf-9377-00aa003b7a11}\InprocServer32’;
{ Returns full path to hhctrl.ocx.
  Returns empty string if file or registry entry not found.}
function GetPathToHHCtrlOCX: string;
var Reg: TRegistry;
begin
  result := ‘’;  //default return
  Reg := TRegistry.Create;
  Reg.RootKey := HKEY_CLASSES_ROOT;
  if Reg.OpenKeyReadOnly(hhPathRegKey) then
  begin
    result := Reg.ReadString(‘’);  //default value
    Reg.CloseKey;
    if (result <> ‘’) and (not FileExists(result)) then
      result := ‘’;
  end;
  Reg.Free;
end;
{setup HTML Help API function interface
 HHCtrlHandle = 0 if API functions not available }
procedure LoadHtmlHelp;
var OcxPath: string;
begin
  if HHCtrlHandle = 0 then   //not already loaded
  begin
    OcxPath := GetPathToHHCtrlOCX;
    if (OcxPath <> ‘’) and FileExists(OcxPath) then
    begin
      HHCtrlHandle := LoadLibrary(PChar(OcxPath));
      if HHCtrlHandle <> 0 then
      begin
        @HtmlHelpA := GetProcAddress(HHCtrlHandle, ‘HtmlHelpA’);
        @HtmlHelpW := GetProcAddress(HHCtrlHandle, ‘HtmlHelpW’);
        @HtmlHelp := GetProcAddress(HHCtrlHandle, ‘HtmlHelpA’);
      end;
    end;
  end;
end;
procedure UnloadHtmlHelp;
begin
  if HHCtrlHandle <> 0 then
  begin
    FreeLibrary(HHCtrlHandle);
    HHCtrlHandle := 0;
  end;
end;
```

The HtmlHelp() API function

Since we now have access to the API, let's look at the HtmlHelp() function, which gives us access to all the API commands. You can find full documentation on how to use the function in the HTML Help Workshop online help, (section "HTML Help References", subsection "HTML Help API").

While we are looking at the HtmlHelp() function and its parameters, I will draw attention to a few problems we have using it under Delphi.

```
function HtmlHelp(hwndCaller: HWND; pszFile: PChar;
    uCommand: UInt; dwData: DWORD): HWND;
```

hwndCaller

Set this to zero. Some programmers use GetDesktopWindow() but zero will do fine. To make the help window stay on top of the application, set hwndCaller to the handle of the owner window. Some advanced HTML Help commands also requires a window handle so that the HTML Help API knows where to send notification messages. That brings us to problem one.

Problem 1. Run example 1 or 2 and open help in stay on top mode, then open a modal form and click on the help window. At this stage the Delphi application gets confused and the modal form falls behind the main form. Note that this works fine under C++ and VB. The problem is probably due to the special way in which Delphi manages forms. I recommend that you stick with zero or GetDesktopWindow. If you do need to specify an owner window, then you should close the help window before opening modal windows.

Problem 2. The first problem is accentuated under HTML Help 1.0, where the help window actually becomes locked between the main form and the modal form. This happens even when using zero or GetDesktopWindow. I recommend that you stick with HH 1.2 and above.

pszFile

This parameter depends on what command you specify in the uCommand parameter. Normally it would specify the full path to the CHM help file. Additionally it can also contain topic and window information. Remember to typecast Pascal strings to PChar. Examples follow.

uCommand

Specify an API command. Here are the three most commonly used commands.

Command	Description
HH_DISPLAY_TOPIC	Opens a help file. Optionally you can specify a help topic and help window. Set dwData to zero.
HH_HELP_CONTEXT	Opens a help topic using a help context ID. Optionally you can specify a help window. Set pszFile to the CHM file. Set dwData to the help context ID.
HH_CLOSE_ALL	Closes all help windows that have been opened by the application. Set all other parameters to zero or nil.

dwData

Set to a value appropriate to the command.

Function return

Depends on the command. Normally returns the handle of the help window.

Using a help window's handle, your application can use Windows API functions to hide, show, position, resize, minimize, maximize, restore, or close a help window. For example, you could tile the help and application windows using Windows API function SetWindowPos(). Another example is making the help window minimize and restore with the application window. Code Example 8 contains these and several other examples.

But what if the user closes the help window? Before using a handle you should check that it is still valid by calling Windows API function IsWindow(hWnd).

Examples using HtmlHelp()

Here are some examples of using the HtmlHelp() API function.

```
var helpfile: string = 'c:\test\help.chm'; h: HWND;
```

1. **Open the help file.**

   h := HtmlHelp(0, PChar(helpfile), HH_DISPLAY_TOPIC, 0);

2. **Open the help file at a topic called intro.htm.**

   h := HtmlHelp(0, PChar(helpfile + '::/intro.htm'), HH_DISPLAY_TOPIC, 0);

3. **Open the help file at a topic called intro.htm in folder html.**

   h := HtmlHelp(0, PChar(helpfile + '::/html/intro.htm'), HH_DISPLAY_TOPIC, 0);

4. **Open the help file at a topic called intro.htm in the help window main.**

   h := HtmlHelp(0, PChar(helpfile + '::/intro.htm>main'), HH_DISPLAY_TOPIC, 0);

5. **Open the help file using context help ID = 1001.**

   h := HtmlHelp(0, PChar(helpfile), HH_HELP_CONTEXT, 1001);

6. **Open the help file using context help ID = 1001 and in the help window main.**

   h := HtmlHelp(0, PChar(helpfile + '>main'), HH_HELP_CONTEXT, 1001);

7. **Close all help windows opened by the application**

   h := HtmlHelp(0, nil, HH_CLOSE_ALL, 0);

Code Example 1 & 2

The first two Delphi code examples demonstrate the three basic API commands we have talked about so far. The examples programs contain a modal dialog so you can experience the Delphi HTML Help modal window problem first hand.

Example 1 and 2 are actually the same, except that example 1 uses static loading and example 2 uses dynamic loading.

Basic help commands

Most of the time we simply want our application to open a help window at a particular topic. We can specify the topic by either supplying a path to the internal file in the CHM help file, or by using an integer help context ID. When using context IDs the relationship between ID and help topic is defined on the help file side using a help-authoring program such as Microsoft HTML Help Workshop.

Another aspect of displaying help is the configuration of the help window. You would normally setup one or more “window types” to use. This is normally done in the help file, but with some extra effort, you can also set up and control windows from the application side. Once a window type has been defined, you use its name in the API call to display the help topic. Window types control all aspects of the help window: what navigation tabs to show; what toolbar buttons to show; whether the window should open with the same size and position it had when it was last closed; and whether the window has the stay-on-top window style. We will look at how to program window types later.

Each topic in a compressed help (.chm) file is actually a separate HTML file. You can think of the CHM file as a mini file system. The help author compiles a collection of HTML files into a single compressed help file using some HTML Help authoring tool, such as Microsoft HTML Help Workshop. Microsoft also provides a command line compiler called hhc.exe. The help project (.hhp) file tells the compiler how to compile the help. When we display a help topic, the API opens the specified CHM help file, decompresses the requested topic file, and then loads the file into the topic pane of the help window.

A warning about closing help

With WinHelp, you were required to send the WinHelp API command HELP_QUIT on program termination. This is not required with HTML Help. With HTML Help, when an application shuts down, all help windows opened by that application are automatically terminated by the operating system. A nice feature, but it often causes an access violation. To avoid this problem, you must make sure your application takes care of closing all help windows on shutdown. To do this, send the HTML Help command HH_CLOSE_ALL on your main form’s close event.

```
 procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
 begin
   HtmlHelp(0, nil, HH_CLOSE_ALL, 0);
 end;
```

An alternate and faster way is to send the wm_close message to the HH window. This requires that you have the handle of the HH window. When you display a help topic the HtmlHelp() function returns the handle of the help window. This can be stored in a global variable then used when required. Use the windows API function IsWindow() to first validate the handle.

```
procedure TForm1.FormClose(Sender: TObject; var Action:
  TCloseAction);
begin
  if IsWindow(HHwinHwnd) then
    SendMessage(HHwinHwnd, wm_close, 0, 0);
end;
```

Displaying a help topic

The HH_DISPLAY_TOPIC command is the simplest way to display a help topic in a help file. Notice how two colon characters are used to separate the CHM file path from the internal CHM path to the help topic. This is standard CHM file notation.

```
fn := 'c:\help\myfile.chm::/htm/intro.htm';
 HtmlHelp(0, PChar(fn), HH_DISPLAY_TOPIC, 0);
```

If we supply only the path to the help file, the help window opens using the default topic and default window type as defined in the help file.

```
fn := 'c:\help\myfile.chm';
 HtmlHelp(0, PChar(fn), HH_DISPLAY_TOPIC, 0);
```

To display the topic using a window type called “main,” we would use the following notation. Again we could omit the “::/htm/intro.htm” part to display the default topic.

```
 fn := 'c:\help\myfile.chm::/htm/intro.htm>main';
 HtmlHelp(0, PChar(fn), HH_DISPLAY_TOPIC, 0);
```

I should mention that a fully qualified CHM path must include the pluggable protocol prefix. This is optional for calls to the HTML Help API. The protocol “mk:@MSITStore:” works with IE3 and greater, while “ms-its:”, or “its:” for short, works with IE4 and greater. When would you need to specify a full path? Opening a CHM topic in Internet Explorer is one situation that comes to mind.

For example:

```
 ms-its:c:\help\myfile.chm::/htm/intro.htm>main
```

Displaying help using help context IDs

Using context IDs is another way in which an application can display help topics. In the application each dialog and control is assigned a unique integer help identifier. When F1 is pressed the application sends this identifier, along with the name of the help file to the API, which in turn opens the help window at the topic mapped to the identifier. WinHelp used the same method to display its contextual help.

Here is an example of a typical context help call. We specify the help file to open and a help context ID of 1001.

```
fn := ‘c:\help\myfile.chm’;
HtmlHelp(0, PChar(fn), HH_HELP_CONTEXT, 1001);
```

When the help API receives the HH_HELP_CONTEXT command, it reads the help file’s lookup table to find out what topic is mapped to the help ID 1001 and opens that topic in the help window. If no mapping exists for help ID 1001, then we will get that familiar error "HH_HELP_CONTEXT called without a MAP section". As with the HH_DISPLAY_TOPIC, command we can also specify a window type. This next example opens the topic in a window defined by the window type "main":

```
fn := ‘c:\help\myfile.chm>main’;
HtmlHelp(0, PChar(fn), HH_HELP_CONTEXT, 1001);
```

Defining the context ID (topic mapping)

The most common question asked by programmers is, how do I define the context ID so it maps to a particular help topic? The answer is found in the HTML Help project (.hhp) file. There are two sections you must fill in called MAP and ALIAS. These sections define the relationship between the help IDs and help topics. HTML Help Workshop provides dialogs to set up this relationship but you will find it quicker to use Delphi or notepad to modify the project file.

Here is an example of a typical ALIAS and MAP section. The MAP section defines an identifier and assigns our integer context ID to it. While the ALIAS section associates that identifier with a help topic.

```
[ALIAS]
IDH_HomePage=default.htm
IDH_TestTopic1=htmlfiles\testtopic1.htm
IDH_TestTopic2=mychm::htmlfiles\testtopic2.htm

[MAP]
#define IDH_HomePage 1000
#define IDH_TestTopic1 1001
#define IDH_TestTopic2 1002
```

Alternatively you can use #include statements. At compilation time the #include statements are replaced by the contents of the specified files.

```
[ALIAS]
#include myhelp.ali

[MAP]
#include myhelp.h
```

In ths example, myhelp.ali is a text file containing:

```
IDH_HomePage=default.htm
IDH_TestTopic1=htmlfiles\testtopic1.htm
IDH_TestTopic2=mychm::htmlfiles\testtopic2.htm
```

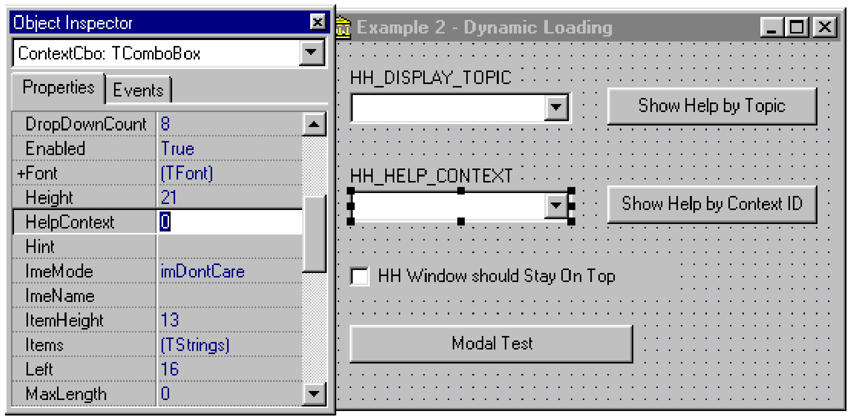
And myhelp.h is a text file containing:

```
#define IDH_HomePage 1000
#define IDH_TestTopic1 1001
#define IDH_TestTopic2 1002
```

Tip: You should add the header and alias files to the [FILES] section of the .hhp file to ensure that the compiler finds the files.

The Delphi side of context help

In Delphi, every form and control has an integer property called “HelpContext.”



ContextCbo: TComboBox

DropDownCount	8
Enabled	True
+Font	(TFont)
Height	21
HelpContext	0
Hint	
ImeMode	imDontCare
ImeName	
ItemHeight	13
Items	(TStrings)
Left	16
MaxLength	0

Figure 1. Delphi Object Inspector shows the selected controls HelpContext property.

When F1 is pressed, Delphi grabs the HelpContext value of the control that has the focus. If this value is zero then Delphi iterates down though the layers of parent controls, right down to the form level. Delphi takes the first non-zero HelpContext value found and makes the call to the WinHelp API for you using that help context ID. If only zero context IDs are found then the help call is not made. Refer to the Delphi online help if you need to know more about how this all works.

We now have a problem because Delphi is setup to use the WinHelp API but we want to use the HTML Help API.

Moving from WinHelp to HTML Help

Suppose your application has more than a hundred dialogs. Each dialog uses field-sensitive help. Converting it to HTML Help will be a nightmare, right? Wrong! Delphi to the rescue. You can convert your entire Delphi application over to HTML Help using a very small amount of code.

Hooking help calls

Read the Delphi online help for the OnHelp event. Using the Application.Onhelp we can write an event handler that will trap all Delphi help events, for our entire application. Once in the handler we can divert each call to the WinHelp API to the equivalent call in the HTML Help API.

The following code shows you how to do it.

```
type
  TForm1 = class(TForm)
    ..
    procedure HelpBtnClick(Sender: TObject);
  private
    FOldHelpEvent: THelpEvent;
    function HelpHook(Command : Word; Data : Longint;
      Var CallHelp : Boolean) : Boolean;
  public
    mHelpFile: String;
  end;
var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {Hook in our help}
  FOldHelpEvent := Application.OnHelp;
```

```
   Application.OnHelp := HelpHook;  //OnHelp now points to our
function
   {Path to chm help file}
   mHelpFile := ExtractFilePath(ParamStr(0)) + 'help.chm';
 end;

 procedure TForm1.FormDestroy(Sender: TObject);
 begin
   {Unhook our help}
   Application.OnHelp := FOldHelpEvent;
 end;

 function TForm1.HelpHook(Command : Word; Data : Longint;
   Var CallHelp : Boolean) : Boolean;
 begin
    if (Command in [Help_Context, Help_ContextPopup]) then
    begin
      {divert to HTML Help Context call}
      HtmlHelp(0, PChar(mHelpFile), HH_HELP_CONTEXT, Data);

      {VCL should not call WinHelp}
      CallHelp := false;
    end;
 end;

 procedure TForm1.HelpBtnClick(Sender: TObject);
 begin
   Application.HelpContext(Self.HelpContext);
 end;
```

Let's walk through the code snippet above. We have created an event handler called HelpHook(). We assign our function HelpHook() to Application.OnHelp in the OnCreate event of our main form. All Delphi HelpContext and the HelpJump methods will now trigger the HelpHook() handler which converts the WinHelp context help calls to HTML Help. The Data value contains our Help Context ID. We restore OnHelp in the form's OnDestroy() event.

Even the help call under the help button will pass though our handler. The help button click event calls Application.HelpContext(HelpContext) which also gets diverted to the HTML Help API via HelpHook().

To find the possible values of the Command and Data parameters, search for WinHelp in the Win32 Developer's Reference Help (Win32.HLP) file, which explains the WinHelp API. The possible values for the Data parameter depend upon the value of the Command parameter.

Code example 3

Code Example 3 uses the above code. Run the compiled executable, tab to any control and press F1 to open HTML Help online help.

Code example 12

The Delphi 6 release has broken the OnHelp event. Only a few help events are not caught by the OnHelp event. Fortunately we can program the new Delphi 6 Help Manager to catch these lost events and pass them correctly onto either Form.OnHelp or Application.OnHelp. The fix is easy to implement. Simple USE the supplied unit “D6OnHelpFix.pas” anywhere in your project. Example 12 demonstrates this fix. Try commenting out “D6OnHelpFix” from the USES section and you will see that half of the help events stop firing.

Programming “What’s This” help

The Delphi code above, that hooks the help calls, will not be enough if you required field level help also know as “What’s This” help.

“What’s This” help is something we see in most Microsoft applications. You click the question mark buttons in window’s title bar, the mouse cursor changes to a pointer with a question mark, you click on a control and you get a popup containing brief help on that control. A keyboard user would tab to the control and press the F1 key to open the popup help.

It takes a lot of work to setup a large application with field level contextual help (Personally I don’t think that most users really gain much from having field sensitive help. Help provided at the page or dialog level can provide a more unified help solution for the user. But that’s just my opinion).

Lets assume that our application requires field-sensitive help.

The “What’s This” buttons

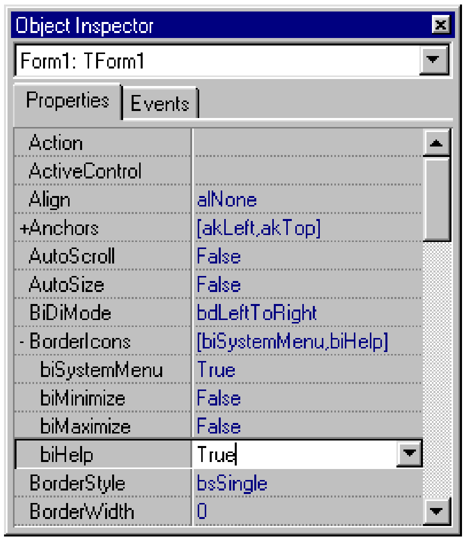
To enable the “Whats This” Help_ContextPopup messages you must set Self.BorderIcons := Self.BorderIcons + [biHelp];

Do this in the FormCreate() or Object Inspector.

Note: The What’s This button in the title bar will not show if you have minimize or maximize buttons.

Delphi provides limited support for What’s This help by placing the question mark icon in the form’s title bar for us. When you click the icon, the cursor changes to crHelp but everything else requires work from us.

To enable the title bar help icon, set the form’s BorderIcons property to include biHelp. The button will be seen at run-time only if BorderStyle is set to bsDialog or biMinimize, and biMaximize id excluded.



Action	
ActiveControl	
Align	alNone
+Anchors	[akLeft,akTop]
AutoScroll	False
AutoSize	False
BiDiMode	bdLeftToRight
-BorderIcons	[biSystemMenu,biHelp]
biSystemMenu	True
biMinimize	False
biMaximize	False
biHelp	True
BorderStyle	bsSingle
BorderWidth	0

Figure 2. Delphi Object Inspector showing the Form's BorderIcons and Border Style properties.

Most applications will also require a "What's This" toolbar button and help menu item. To accomplish this, call the following code. Here is my "What's This" toolbar button click event code.

```
procedure TForm1.WhatsThisToolButtonClick(Sender: TObject);
 begin
   DefWindowProc(handle, WM_SYSCOMMAND, SC_CONTEXTHELP, 0);
 end;
```

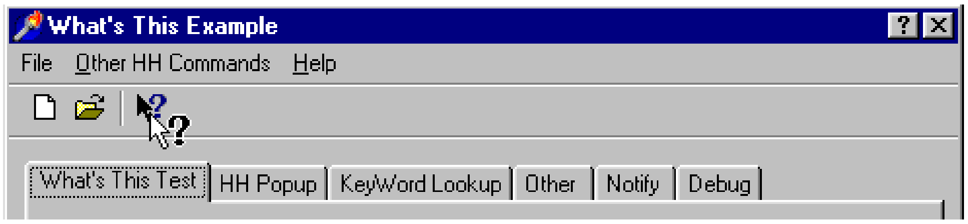


Figure 3. Click the question mark icon to enter What's This help mode.

Catching help events

Just as before, we will use Delphi's Application.OnHelp event to trap all help calls to the WinHelp API.

```
{All application help calls to help come here}
function TForm1.HelpHook(Command: Word; Data: Longint;
  Var CallHelp: Boolean) : Boolean;
begin
  CallHelp := FALSE; //override default handling

  case Command of
    HELP_CONTEXT: //help button - data = helpcontext ID
      HtmlHelp(0, PChar(mHelpFile), HH_HELP_CONTEXT, Data);
    HELP_SETPOPUP_POS: //data = x,y pos for popup
      _pt := SmallPointToPoint(TSmallPoint(Data));

    HELP_CONTEXTPOPUP: //F1 help - data = helpcontext ID
      HH_ShowPopupHelp2(Self, Data, _pt);

    else
      CallHelp := TRUE;  //default handling
  end;
  result := TRUE;
end; { THookHelpSystem.HelpHook }
```

Look at the above code. Calls like Application.HelpContext(1000) used in the help button click event all come through the Help_Context case. Nothing new so far.

When a "What's This" cursor is clicked on a control or F1 is pressed while the control has the focus, Delphi sends two help commands. The first is HELP_SETPOPUP_POS with data containing the x,y position of the control as a TSmallPoint. The second is HELP_CONTEXTPOPUP with data containing the HelpContext ID assigned to the control. If we store the x,y values in a global TPoint variable we can use the control coordinates later when the HELP_CONTEXTPOPUP command arrives. Finally HH_ShowPopupHelp2() function contains code to display the HTML Help popup. We will talk about this function later on.

To understand how Delphi manages the help events, checkout the Delphi source code in Forms.pas. All WM_HELP events come through TCustomForm.WMHelp. Notice that if biHelp border icon is set, two help commands are called. The first contains the x,y coordinates of the control, the second contains the controls helpcontext ID.

```
if (biHelp in BorderIcons) then
begin
  Application.HelpCommand(HELP_SETPOPUP_POS, Longint(Pt));
  Application.HelpCommand(HELP_CONTEXTPOPUP, ContextID);
end
else
  Application.HelpContext(ContextID);
```

We now have all our help events coming through a single function. We have the controls x,y coordinates and its helpcontext ID. All we need to do now is make the call to the HTML Help API to display a popup window.

Code example 4

You can see the finished result by running the Example 4 demo. Click the question mark then click one of the controls on the first page.

HTML Help popups

The HTML Help API provides two popup commands. Note that HH popups are text only at the moment, although some third party rich HTML solutions are starting to surface.

- The HH_TP_HELP_CONTEXTMENU command creates a popup using text from a text file in a chm help file. You pass the handle of the control and the controls help topic ID that is declared in a text file in a chm help file.

  This method is not too useful to us. It offers no control over popup attributes. It does not allow line breaks. It requires the handle of the control, which we do not have so far. Sure we could get the handle with some work, but then why fight Delphi.

  Note that there is no functional difference between the HH_TP_HELP_CONTEXTMENU and the HH_TP_HELP_WM_HELP commands.

- The HH_DISPLAY_TEXT_POPUP command offers the most flexibility. You can set the color attributes and popup margins. The popup text can be specified in three different ways: as an explicit text string, as a text string based on a resource ID and as a topic ID that is declared in a text file in a chm help file.

For completeness I'm going to quickly cover all these methods of displaying a text popup. All these examples can be found in Example 4.

Defining text popups in a CHM

Both API commands can display text defined in a CHM help file so let's see how to do this first. Here is an example. Create a text file called cdhelp.txt and type in the following popup window topics.

<file: cdhelp.txt>

```
.Topic 99
This is my test popup text with ID=99
The source lives in “help.chm::/cshelp.txt”

.Topic IDH_POPUP_NEWBTN
Click new to New to create a new file.

.Topic IDH_POPUP_OPENBTN
Click new to New to create a new file.
```

You can write the context ID number directly into the text file as seen above with Topic 99. Or you can define identifiers in a separate C++ header file as I have done with the other 2 declarations.

<file: cdhelp.h>

```
#define IDH_POPUP_NEWBTN        61
#define IDH_POPUP_OPENBTN       62
```

Lasty we need to add the following lines to our help project (.hhp) file and compile it to a CHM file. Here is my help project file.

<file:help.hhp>

```
[MAP]
#include cshelp.h

[TEXT POPUPS]
cshelp.h
cshelp.txt

[FILES]
cshelp.h
cshelp.txt
```

Tip: Some people have found that including the name of the text and header files in the [FILES] section fixes a bug where the mapping information sometimes cannot be found by the chm.

Popups using HH_TP_HELP_CONTEXTMENU

Now to the API call HH_TP_HELP_CONTEXTMENU. This API command requires that we pass an array of DWords containing control handle and topic ID pairs, terminated by two trailing zeros. We call the API, using the handle of the control that you are requesting help for. The API will look up the ID in the DWord array and display the popup over the control. The second parameter of the HtmlHelp() call is the path to the topic text file we created before.

```
var dwIDs: array[0..3] of DWord;

dwIDs[0] := ShowHHPopupBtn.Handle;
dwIDs[1] := 99;
dwIDs[2] := 0;
dwIDs[3] := 0;

HtmlHelp(ShowHHPopupBtn.Handle, PChar('c:\test\help.chm::/
cshelp.txt'),
  HH_TP_HELP_CONTEXTMENU, DWORD(@dwIDs[0]));
```

Popups using HH_DISPLAY_TEXT_POPUP

This is the more powerful of the two API popup commands. We simply fill in a THHPopup structure and make the call.

```
{Show HH Popup using supplied text}
function HH_ShowPopupHelp1(aParent: TForm; Text: String;
  XYPos: TPoint): HWND;
var hhpopup: HH.THHPopup;
begin
  with hhpopup do
  begin
    cbStruct := sizeof(hhpopup);    //sizeof this structure
    hinst := 0;                     //instance handle if
                                 //str res
    idString := 0;                  //zero, resource ID, or
                                        //topic ID
    pszText := PChar(Text);         //text to display if
                                 //idString=0
    pt := XYPos;                    //top center of popup
    clrForeground := COLORREF(-1);  //use -1 for default -
                                 //RGB value
    clrBackground := COLORREF(-1);  //use -1 for default -
                                 //RGB value
    rcMargins  := Rect(-1,-1,-1,-1);//amount of space
                                 //between edges
    pszFont := '';                  //font
  end;
  Result := HtmlHelp(aParent.Handle, nil,
    HH_DISPLAY_TEXT_POPUP, DWORD(@hhpopup));
end;
```

In the HH_ShowPopupHelp1() example above, function parameter aParent is the form that will parent the popup, Text is the text string to display and XYPos holds the x-y coordinates for the popup.

To display a text string based on a resource ID we would use the form below. The strings for example 4 are stored in the application so we simply assign hinst=hInstance and idString=resID. For more information on how to store resource strings in a Delphi application see your Delphi manual.

```
{Show HH Popup using a string resource}
function HH_ShowPopupHelp2(aParent: TForm; resID: Integer;
   XYPos: TPoint): HWND;
var hhpopup: HH.THHPopup;
begin
  with hhpopup do
  begin
    cbStruct := sizeof(hhpopup);    //sizeof this structure
    hinst := hInstance;             //instance handle if
                                        //str res
    idString := resID;              //zero, resource ID, or
                                        //topic ID
    pszText := nil;                 //text to display if
                                        //idString=0
    pt := XYPos;                        //top center of popup
    clrForeground := COLORREF(-1);  //use -1 for default -
                                        //RGB value
    clrBackground := COLORREF(-1);  //use -1 for default -
                                        //RGB value
    rcMargins  := Rect(-1,-1,-1,-1);//amount of space
                                        //between edges
    pszFont := '';                  //font
  end;
  Result := HtmlHelp(aParent.Handle, nil,
    HH_DISPLAY_TEXT_POPUP, DWORD(@hhpopup));
end;
```

Finally the third and last example (below) tells the API to grab the popup text from the text file in a chm, like we did in the HH_TP_HELP_CONTEXTMENU example. Notice the second parameter is now defined as the path to the topic text file in the chm.

```
{Show HH Popup using a string (StringID) from text file in a CHM
 StringID: eg. 99;
 CHMTextFile: eg. _runDir + 'help.chm::/cshelp.txt'}
function HH_ShowPopupHelp3(aParent: TForm; StringID: Integer;
 CHMTextFile: String; XYPos: TPoint): HWND;
var hhpopup: HH.THHPopup;
begin
  with hhpopup do
  begin
    cbStruct := sizeof(hhpopup);    //sizeof this structure
    hinst := 0;                     //no used
    idString := StringID;           //topic number in a
                                        //text file.
    pszText := nil;                 //no used
    pt := XYPos;                    //top center of popup
    clrForeground := COLORREF(-1);  //use -1 for default -
                                        //RGB value
    clrBackground := COLORREF(-1);  //use -1 for default -
                                        //RGB value
    rcMargins  := Rect(-1,-1,-1,-1);//amount of space
                                        //between edges
    pszFont := '';
  end;
  Result := HtmlHelp(aParent.Handle, PChar(CHMTextFile),
    HH_DISPLAY_TEXT_POPUP, DWORD(@hhpopup));
end;
```

Just to wrap up these HH_DISPLAY_TEXT_POPUP popups. The color parameters are either –1 for the default color or an RGB value. The easiest way is to specify a color is to use a three byte hexadecimal number where each byte represents a Blue, a Green and a Red value, $00BBGGRR.

Example of red text:

```
clrForeground := COLORREF($000000FF);
```

Example of blue text:

```
clrForeground := COLORREF($00FF0000);
```

The pszFont parameter allows you to specify a font. Use an empty string for the default font or specify 'facename, point size, char set, BOLD ITALIC UNDERLINE'. You can leave a parameter blank to specify that the default value should be used.

Example: Char set is not specified so the default charset will be used:

```
pszFont := 'MS Sans Serif, 10, , BOLD';
```

More help on popups can be found in the API reference in Workshop online help.

Rich HTML popups

HTML Help 1.x provides plain text popups only. Some third party vendors are developing Rich HTML popups. The best example is KeyHelp, developed by KeyWorks Software, a division of Work Write, Inc.:

http://www.keyworks.net/keyhelp.htm

Other HTML Help Programming Issues

I want to step aside from code examples for a moment and talk about some other programming issues that effect Delphi programmers.

Isolating the application from the API

Some situations may require you to avoid calling the HTML Help API directly and instead open help via a second executable. Why would we want to do this?

- To make the application more robust. As you will soon see, playing host to the HTML Help API can be dangerous. If help falls over it may take your application with it.
- You may have a requirement to keep the help window opened when the executable is closed. Normally all help windows opened by the executable are closed automatically when the application shuts down.

- So we can use Delphi 3 ActiveX in-process servers in our HTML online help.

  Problem 4: Access violations occur when Delphi 3 applications hosts the HH API. This in turn hosts the Internet Explorer WebBrowser control. This in turn hosts a Delphi made in-process ActiveX control or automation server. This problem only occurs with this combination of components.

Let's step aside for a moment to study how DLLs work.

The HTML Help API is an OCX but we use it like a normal DLL. When a process loads a DLL, the system maps the code and data for the DLL into the address space of the process. For 32-bit Windows, DLLs become part of the process that loads the DLL. Any memory allocation calls made by functions in the DLL cause memory to be allocated from the process's address space.

Only one copy of a DLLs code and data is ever kept in memory. Since code and data parts do not change it is safe for the operating system to simply map them into the address space of each process that loads them.

So looking at the bigger picture… we have our application that plays host to the HTML Help API. The HTML Help window itself plays host to the Internet Explorer WebBrowser control, Shdocvw.dll. The WebBrowser control in turn plays host to… well, think about all those things you can run in Internet Explorer. Now think about the number of times your browser crashes in a day. Okay so your online help is plain text and graphics, in which case you have nothing to be nervous about, maybe¿

Delphi 3 COM DLL bug

The last item mentioned is a particularly nasty bug, since ActiveX is the most common method of extending the capabilities of HTML Help topics. Here are some ways to work-around the problem.

- nstead of writing an in-process server (DLL), write an out-of-process server (EXE). The result is the same as far as the user is concerned. You can hide the main window by setting Application.ShowMainForm := FALSE in the main form's FormCreate event.

- Your help topics can call external helper programs using HTML Help shortcut commands. Not as flexible as ActiveX but very simple to implement. See Workshop help for further info on using shortcuts.

- Upgrade to Delphi 4 or 5. Delphi 4 must have all available patches applied to it. These are free to download from the Borland site at http://www.borland.com/

- Write your COM DLL in C++ or VB.
- Launch your online help from a second C++ or VB application.

Launching HTML Help from a second application

If you do choose to launch HTML Help from a second helper application here are some options.

Write a small non-Delphi application that loads the help API and accepts command line parameters to perform the required help commands. The program should be run only when required. The main window should be invisible. Remember that the help windows owned by the application will close automatically when the application closes, so you must keep the executable running until all its help windows are closed. A way to achieve this is to store the window handles returned by API function calls, Implement a timer event and periodically check if the window handle is still valid using windows API function IsWindow().

Mmm. This solution is probably more work than we expected. Never mind there are two programs already available that will do the job. Both are written in C++ and both are free.

- HH.EXE is distributed with HTML Help so you can rely on it being present. It lives in the Windows folder and has a limited number of command-line options. HH.EXE is not single instance, so if the user pressed F1 three times, then three help windows would appear.

Example of opening a help topic using Help ID = 1001

```
HH.EXE -mapid 1001 ms-its:path/help.chm
```

Note: The "—map ID chm" command line became available in HH 1.1b.

Example of opening a help topic using a topic path

```
HH.EXE  ms-its::path/help.chm::/path/topic.htm
HH.EXE  mk:@MSITStore::path/help.chm::/path/topic.htm
```

Note: The mk:@MSITStore protocol works with IE3 and above while ms-its works with IE4 and above. A shorter version of "ms-its" is to just use "its". Actually, the later versions of HH don't even require the protocol prefix.

- KeyHH.EXE is a free program from KeyWorks Software, written by Ralph Walden. KeyHH.EXE supports many more command line options then HH.EXE. I have to say that KeyHH.EXE is an excellent program. I distribute it with most applications I help produce.

  You can get more information about KeyHH.EXE from the KeyWorks web site at:

  http://www.keyworks.net/keyhh.htm

- If you have a copy of MS Visual Basic then David Liske's HTML Help DLL makes creating a VB exe with HH support a snap. The DLL also works with Delphi applications.

  Download this example from:

  http://www.mvps.org/htmlhelpcenter/

Free Delphi code (Woo hoo!)

We have covered a bit of ground. It would be nice to package the code we have so far into a Unit for easy reuse. With the example code is a folder called CommonCode. Here you will find two units I have created that you are free to use.

HH.pas

This unit is a port of the C++ header file "htmlhelp.h". It contains all the HTML Help commands and data structures as well as an interface to the API function HtmlHelp() using dynamic DLL loading. The LoadLibrary() call is made during module initialization.

HH_Funcs.pas

This optional unit contains other HTML Help related functions:

- The THookHelpSystem object wraps the Example 3 code. Create an instance of the object in main form create and free it on main form destroy. You can also specify whether the context help command is called using the help API directly, with HH.EXE or with KeyHH.EXE.

- Functions to parse HTML help paths.

- Functions to check the versions of HH and IE installed.
- Functions to open help via Windows via “HH.EXE” and “KEYHH.EXE”

Documentation for the units can be found in file hh_doc.txt found in the same folder.

The latest version of these units are kept on my web site at:

http://helpware.net/delphi/

Other HH API commands

In this next section we will examine the remaining API commands. We will cover:

- Programming the Contents, Search and Index tabs
- Get Last HTML Help Error command
- Keyword Search command
- Synching Contents Tab command
- Programming HH Windows

Code Example 4

Code Example 4 demonstrates the above topics. I don’t want to fill the chapter with code, so I will refer to the source code in “Example 4\Unit1.pas” as we go along. The executable is called “Example 4\WhatsThis.exe”

The Contents, Search and Index tabs

Often applications have commands in the help menu that allow the user to quickly jump to the Contents, Index or Search navigation tabs on the help window. Lets see how HTML Help does this. Note that there is currently no API command available for selecting the favorites tabs.

HH_DISPLAY_TOC

This help command opens the help at the contents tab.

```
s := ‘c:\test\help.chm’;
HtmlHelp(0, PChar(s), HH_DISPLAY_TOC, 0);
```

Optionally, you can open help at a specific topic. But this only works if the help window is originally closed.

```
s := 'c:\test\help.chm::/htmlfiles\testtopic2.htm';
HtmlHelp(0, PChar(s), HH_DISPLAY_TOC, 0);
```

HH_DISPLAY_INDEX

This help command opens the help at the index tab.

```
HtmlHelp(0, PChar(mHelpFile), HH_DISPLAY_INDEX, 0);
```

Optionally you can also perform a keyword search for a keyword.

```
s := 'Test Topic 2';  //Key word
HtmlHelp(0, PChar(mHelpFile), HH_DISPLAY_INDEX, DWord(PChar(s)));
```

HH_DISPLAY_SEARCH

This command opens help at the search tab. The command requires that we fill in a THHFtsQuery record. Optionally you can perform a search, but this appears to be broken under HH 1.x.

```
var q: THHFtsQuery;
begin
  q.cbStruct        := sizeof(q); //Sizeof structure in
                                     //bytes.
  q.fUniCodeStrings := FALSE;     //all strings are
                                     //unicode.
  q.pszSearchQuery  := 'test';    // the search query text
                                     //string.
  q.iProximity      := HH_FTS_DEFAULT_PROXIMITY;//Word
                                                     //proximity
  q.fStemmedSearch   := FALSE;    //TRUE for Stemmed
                                     //Search only.
  q.fTitleOnly       := FALSE;    // TRUE for Title search
                                     //only.
  q.fExecute         := TRUE;     // TRUE to initiate the
                                     //search.
  q.pszWindow        := nil;      //Window to display in
                                     //HtmlHelp(0,
                                     //PChar(mHelpFile),
                                     //HH_DISPLAY_SEARCH,
                                     //DWORD(@q));
end;
```

The above opens help at the search tab and performs a search on the word 'test'. The iProximity member must be always set to HH_FTS_DEFAULT_PROXIMITY.

To simply open at the search tab with no search, clear the following three fields.

```
q.pszSearchQuery   := nil;   // String containing the
                                // search query.
q.fExecute         := FALSE; // TRUE to initiate the
                                // search.
q.pszWindow        := nil;   // Window to display in
```

Get Last HTML Help error

The HH_GET_LAST_ERROR command returns the last error reported by the HTML Help API. This is very handy for debugging. The Example 4 code calls the following function after every call to the API and if an error has occurred, writes the error description string to the debug log on the last page tab. The function below returns an empty string if no error description was available. Note that the HH_GET_LAST_ERROR command does not clear the error information. Also note that the description string is a BSTR and must be freed using the SysFreeString() function.

```
{HH Get Last Error command — returns '' if no error available}
function TForm1.GetLastHHError: String;
var lasterror: THHLastError; h: HWND;
begin
  result := '';
  lasterror.cbStruct := sizeof(lasterror); //sizeof this
                                                //structure
  lasterror.hr := 0;                       //returns the
                                                //error code
  lasterror.description := nil;            //Unicode
                                                //descriptor str
  h := HtmlHelp(Self.Handle, nil, HH_GET_LAST_ERROR,
    DWORD(@lasterror));
  if (h <> 0) then // Make sure that HH_GET_LAST_ERROR
                     // succeeded.
  begin
    //Only report error if failure and error description
    //available
    if (lasterror.hr < 0) and (lasterror.description <> nil) then
    begin
      result := WideCharToString(lasterror.description);
      SysFreeString(lasterror.description);  //BSTR must be freed
    end;
  end;
end;
```

Keyword searches

Both HH_ALINK_LOOKUP and HH_KEYWORD_LOOKUP commands use the THHAKLink structure to search for ALinks and KLinks respectively. If the search for a keyword is successful, a dialog appears listing the matching ALinks or KLinks topics. If only one result is found the dialog is skipped and the topic is immediately loaded. The following code searches for the keyword 'Agent'.

```
var link: THHAKLink;
begin
  link.cbStruct := sizeof(link);
  link.fReserved := FALSE;       //must be FALSE (really!)
  link.pszKeywords := 'Agent';   //semi-colon separated
                                    //keywords
  link.pszUrl := nil;            //URL if no keywords
                                    //found
  link.pszMsgText := nil;        //Msg box text if no
                                    //keyword match
  link.pszMsgTitle := nil;       //Msg box title text
  link.pszWindow := nil;         //Window to display URL in
  link.fIndexOnFail := TRUE;     //Displays index if
                                    //failed
```

For ALinks call:

```
   HtmlHelp(0, PChar(chmFile), HH_ALINK_LOOKUP,
DWORD(@link))
```

For KLinks call:

```
  HtmlHelp(0, PChar(chmFile), HH_KEYWORD_LOOKUP,
DWORD(@link));
```

There are a few variations to the call we can make. We can get the API to display a message box when the search fails:

```
 s1 := 'KeyWord Search';
 s2 := 'Keyword search failed.' + #13#10 + 'Retype and try
again!';
 link.pszMsgTitle := PChar(s1);
 link.pszMsgText := PChar(s2);
 link.fIndexOnFail := FALSE;      // Set FALSE to see the
                                     // message
```

We can get the API to display a URL when the search fails, say a topic in a CHM. We can also specify the window type to use.

```
s1 := 'htmlfiles\FileNotFound.htm';
 s2 := 'window2';
 link.pszUrl := PChar(s1);    // URL to jump to if keyword
                                 // not found
 link.pszWindow := PChar(s2); // Window to display URL in
 link.fIndexOnFail := FALSE;  //  Set to FALSE to see the
                                 // message
```

Synching the Contents tab

The HH_SYNC command can be used to select an item in the table of contents. Note that this does not load the topic. It simply selects the topic name in the table of contents. In the call you must specify a window type with the chm file, in this case 'main' and the name of the topic to select.

```
fn := mHelpFile + '>main';
topic := mHelpFile + '::/mytopic.htm';
HtmlHelp(0, PChar(fn), HH_SYNC, DWord(PChar(topic)));
```

Getting the window handle

Code examples 8 and 9 demonstrate how you can manipulate HTML Help windows using the window handle and appropriate window API calls. The help window handle is returned by the HtmlHelp() API function when you display a window. Another way of finding the help window handle is using the HH API command HH_GET_WIN_HANDLE.

HH_GET_WIN_HANDLE

This command will retrieve the handle of a help window. The call returns zero if the help window has not yet been created. Otherwise it returns the handle of the requested window. The call requires the path to the CHM help file and the name of the window type to find (in this case "main").

```
 var h: HWND; s: string;
 begin
   s := 'c:\test\help.chm';
   h := HtmlHelp(0, PChar(s), HH_GET_WIN_HANDLE,
WORD(PChar('main')));
```

Programming HH window types

Here's where things start to get more complex. I mentioned before that we could program our own window types. The HTML Help API provides commands to read the window type information of a help window as well as update or create a window type.

HH_GET_WIN_TYPE

This command returns a pointer to a THHWinType window type structure owned by the HTML Help API. This data should be considered read-only. See the HH_SET_WIN_TYPE command below for more information

For example code please see example 4.

```
var pwintype: PHHWinType; fn: String;  h: Integer;
```

```
begin
  fn := mHelpFile + '>main';
  h := HtmlHelp(0, PChar(fn), HH_GET_WIN_TYPE, DWORD(@pwintype));

  if h = 0 then
    ShowMessage('** Window help.chm>main not opened.')
  else if h = -1 then
    ShowMessage('** Window help.chm>main has not been defined.')
  else if h < 0 then
    ShowMessage('** Call failed. ' + GetLastHHError)
  else
  with pwintype^ do
  begin
    writeln(f, 'pszType', pszType);  //IN/OUT: Name of
                                         //window
    writeln(f, inttostr('fsWinProperties'); //IN/OUT:
                                                 //Properties
    ...
```

HH_SET_WIN_TYPE

This command can be used to create or modify a window definition. Note: It is a lot simpler to set up window definitions using workshop.

The following code creates a window type called 'mywindow' for the CHM file specified. It has no toolbar or navigational panes. Some window type items cannot be modified, or will not change until the window is closed and reopened.

To modify an existing windows type it may be useful to GET the window type information before you SET the window type information. Don't modify the HTML Help API's data directly, instead make a deep copy of the data structure and use this to SET the window type.

Looking at the code below we have filled the THHWinType structure with zeros, then filled in the record members. We set pszType to be the name of the window type. This is the window type name we will use in future calls to the HH API. Set pszCaption to specify what text to display in the help window's title bar. fsValidMembers tells the API which members we are setting. In this example we are only setting three members, fsWinProperties, rcWindowPos and nShowState.

```
var wintypedef: THHWinType; h: HWND;
    winName: string;
begin
  winName := 'mywindow';                   //window name
  fillChar(wintypedef, sizeof(wintypedef), 0);  //fill with
                                                     //zeros
  with wintypedef do
  begin
    cbStruct := sizeof(wintypedef); // IN: size of this
                                       // structure
    pszType := PChar(winName);      // IN/OUT: Name of a
                                       // window
    pszCaption := 'My Window';       // IN/OUT: Caption
    fUniCodeStrings := FALSE;        // IN/OUT: UNICODE
                                         // strings
    fsValidMembers :=                // IN: Bit flag of
                                         // valid members
      HHWIN_PARAM_PROPERTIES    or  // valid
                                         // fsWinProperties
      HHWIN_PARAM_RECT          or  // valid rcWindowPos
      HHWIN_PARAM_SHOWSTATE;        // valid nShowState
    fsWinProperties :=               // IN/OUT: Properties/
                                         // attributes
      HHWIN_PROP_ONTOP          or  // Top-most window
      HHWIN_PROP_NO_TOOLBAR;        // Don't display a
                                         // toolbar
    rcWindowPos := Rect(5, 5, 330, 330);
                  // IN: Starting position, OUT: current position
    nShowState := SW_SHOW;            // IN: show state SW_*
  end; //with
  {API call to create/modify the window type}
  h := HtmlHelp(0, PChar(mHelpFile), HH_SET_WIN_TYPE,
    DWORD(@wintypedef));
```

We can optionally create a global window type, by setting the second parameter of the API call to nil. Any CHM help file opened by the current process can use a global window type. This is especially useful for merged help systems where you want several help files to share a single help window. This is the same as defining a window type with workshop and prefixing $global_ to the window type name.

```
{create a global window type}
h := HtmlHelp(0, nil, HH_SET_WIN_TYPE, DWORD(@wintypedef));
```

More about window types

All windows types in HTML Help 1.0 and 1.1 were always global for a process. Different processes cannot access each other's window type information. HTML Help 1.2 changed the rules and attached each window type to a specific chm file unless it was created as a global window type.

Check out the HH_WINTYPE structure, which we call THHWinType for Delphi, in hh.pas. Each item contains a comment of either IN or OUT or both IN & OUT. IN items may be used by the HH_SET_WIN_TYPE command, while OUT items can be used with HH_GET_WIN_TYPE command.

Some items need further explanation. Also check the hh.pas comments for further detail.

Structure	Description
cbStruct	IN. Set to the size of the THHWinType data structure.
fUniCodeStrings	IN. This setting is independent of whether you are calling the API using HtmlHelpA or HtmlHelpW. If set TRUE the API treats all strings as Unicode.
pszType	IN/OUT: The name of the window type.
fsValidMembers	IN. This bit field allows you specify which members you want to set. For example HHWIN_PARAM_PROPERTIES bit means that the fsWinProperties item is being set. Check out the comments in the header file to see the relationship between bit flags and structure members.
fsWinProperties	IN/OUT. HHWIN_PROP_* bit-flags. Include HHWIN_PROP_TRI_PANE to create a tri-plain window.
pszCaption	IN/OUT. The caption text for the window title bar.
dwStyles	IN/OUT: Window styles bit-flags. Example: WS_VISIBLE. See the Win32 API help.
dwExStyles	IN/OUT: Window extended styles bit-flags. Example: WS_EX_APPWINDOW. See the Win32 API help.
rcWindowPos	IN: Starting position, OUT: current position
nShowState	IN: Show state. Example: SW_SHOW. See the Win32 API help.
hwndHelp	OUT: Help window handle. Same handle returned by HtmlHelp().
hwndCaller	OUT: handle of caller window.
paInfoTypes	IN: Not Used. Set to nil.

These items are used if the window is a tri-plane window.

Structure	Description
hwndToolBar	OUT: Handle of the toolbar window pane.
hwndNavigation	OUT: Handle of the navigation window pane.
hwndHTML	OUT: Handle of the topic window pane.
iNavWidth	IN/OUT: Width of navigation window pane.
rcHTML	OUT: Size and position of the topic window pane.

Structure	Description
pszToc	IN: URL of the table of contents file (.hhc).
pszIndex	IN: URL of the index file (.hhk).
pszFile	IN: URL of the HTML file.
pszHome	IN/OUT: URL displayed when Home button is clicked.
fsToolBarFlags	IN: Toolbar Bit-flags controlling the appearance. See HHWIN_BUTTON_* in the header file.
fNotExpanded	IN: Hide or Show the navigation pane, OUT: current state.
curNavType	Not Used.
tabpos	IN/OUT: Navigation pane tab layout. HHWIN_NAVTAB_TOP, HHWIN_NAVTAB_LEFT, or HHWIN_NAVTAB_BOTTOM
idNotify	IN: HTML Help passes this ID to the applications WM_NOTIFY event if enabled.
tabOrder	Not Used.
cHistory	Not Used.
pszJump1	IN/OUT: Text for HHWIN_BUTTON_JUMP1
pszJump2	IN/OUT: Text for HHWIN_BUTTON_JUMP2
pszUrlJump1	IN/OUT: URL for HHWIN_BUTTON_JUMP1
pszUrlJump2	IN/OUT: URL for HHWIN_BUTTON_JUMP2
rcMinSize	IN/OUT: Not used. Minimum size for the window.
cbInfoTypes	IN: Not Used. Set to zero.
pszCustomTabs	Not Used.

Bridging the gap between help and the application

As online help evolves, we see the boundaries between help and application blur more and more. Sometimes it's hard to see where the application ends and the help begins.

There are several ways we can get the application and help to work more closely together.

- Script & Java Applets

  The most common way to extend the capabilities of help is to write JavaScript & VBScript. If you have a Java compiler like Borland's J-Builder then Java Applets can provide even more functionality. You will find plenty of good script examples and tutorials on the web. The script reference help for writing MS JScript and VBScript is a good starting point. It can be download for free from the MS web site: http://msdn.microsoft.com/scripting/

- ActiveX

  The most powerful way to extend help is to write ActiveX. Using script the HTML document can call any Delphi ActiveX function you care to write. The function may do some action, return a value, display a dialog or create a new HTML topic on the fly. Note that in WinHelp we created DLLs to extend help. In HTML Help we must create ActiveX. A simple example follows later.

- HTML Help Shortcuts

  HTML Help Workshop gives you the ability to run external executables from a help topic. See Workshop help for more information.

- Notification Messages

  You can make a HTML window send a notification message to an application whenever the topic changes or buttons are clicked. More on Notification message later.

- Help Window Handle

  We have seen a number of different ways in which we can obtain the handle to a help window. Once you obtain a window handle, you can use windows API functions to move, resize, close, hide, show, minimize and maximize the window. Example 8 demonstrates how to manipulate a window using its handle.

- HH_GET_WIN_TYPE API command

  We covered this above. It's a good way for the application to see what is happening to a help window. Note that if you use Notification messages then you will receive a pointer to the THHWinType data structure when the HHN_TRACK notification message arrives.

- Training Cards

  A very useful but little used command. Basically, in Workshop you setup a help topic to send a WM_TCARD message to the application via some action in a help topic. You can send a string or integer with the message. The application receives the message and responds in whatever way you want. An example follows.

- Embedding TWebBrowser control

  Merge the help right into the application. This is exactly what HTML Help window does. The Topic pane is simply a TWebBrowser control. You get events from the browser and more or less total control over its contents. More on this later.

- Embedding HTML Help window

  You can embed the entire help window if you want. An example of this is seen in HH Workshop. Click the help toolbar button and Workshop embeds a HH window, which has just the contents and toolbar panes. An example follows.

HTML Help notification messages

When you create a window type using the HH_SET_WIN_TYPE command, you can specify that the help window should send the application WM_NOTIFY messages. There are three types of messages that are sent to the application:

Message	**Description**
HHN_NAVCOMPLETE	Sent when the user navigates to a new topic. The URL is sent with the message.
HHN_TRACK	Sent when a user clicks a button on the toolbar. The message contains the current topic URL, button ID and a pointer to the windows THHWinType data structure. The message is sent just before the action is taken.
HHN_WINDOW_CREATE	Sent just before a help window is created.

Enabling notification messages

When the help window is created using HH_SET_WIN_TYPE we must set up the following THHWinType items.

Message	Description
idNotify	Set this to some unique ID. When the application receives a notification message it will know which help window sent the message by this ID.
fsValidMembers	Make sure the bit-flag HHWIN_PARAM_PROPERTIES is included. This enables the fsWinProperties member of the data structure.
fsWinProperties	Make sure the bit-flag HHWIN_PROP_TRACKING is included to enable tracking notification messages.

Once you have set up your data structure, call HtmlHelp() using the HH_SET_WIN_TYPE command to create the window type. See the section on HH_SET_WIN_TYPE above for more information.

```
  h := HtmlHelp(0, nil, HH_SET_WIN_TYPE,  DWORD(@wintypedef));
```

When you open your help window, you must specify the handle of the window that will receive the WM_NOTIFY messages.

```
  fn := ‘c:\test\help.chm::/Agent\Genie.htm>mywin’;
  HtmlHelp(Self.handle, PChar(fn), HH_DISPLAY_TOPIC, 0);
```

To see notification messages in action, see the Notify page tab of the Example 4 sample application. Check out the code under the “Start Notify Test” button.

Receiving notification messages

To set up our Delphi application to receive WM_NOTIFY messages from the help window, place the following code in your main form class.

```
 type
   TForm1 = class(TForm)
     ...
   private
     procedure WMNotify(var Message: TWMNotify); message
WM_NOTIFY;
     ...
   end;
```

Here is the Form1.WMNotify code. Assume that the Notify ID we setup in the THHWinType data structure was 999. When the notify event comes in we are only interested in notifications from ID=999. Pointers to THHNNotify and THHNTrack data structures are used to pull out the data passed in the event.

```
 procedure TForm1.WMNotify(var Message: TWMNotify);
 var s: String; phhn: PHHNNotify; phht: PHHNTrack; pwt:
PHHWinType;
 begin
   with Message do
   begin
      if Self.Showing and (NMHdr.idFrom = 999) then
      begin
        case NMHdr.code of
          HHN_NAVCOMPLETE:
            begin
              phhn := PHHNNotify(NMHdr);
              s := phhn^.pszUrl;
              Memo.Lines.Add('HHN_NAVCOMPLETE - pszUrl = ' + s);
            end;
          HHN_TRACK:
            begin
              phht := PHHNTrack(NMHdr);
              s := phht^.pszCurUrl;
              pwt := phht^.phhWinType;   // window type
                                             // structure
              Memo.Lines.Add('HHN_TRACK - pszCurUrl = ' + s);
              Memo.Lines.Add('          - idAction = '
                + inttostr(phht^.idAction); //HHACT_ value
            end;

          HHN_WINDOW_CREATE:
            begin
              phhn := PHHNNotify(NMHdr);
              s := phhn^.pszUrl;
              Memo.Lines.Add('HHN_WINDOW_CREATE - pszUrl =
%s',[s]));
            end;
        end; //case
      end; //if
   end; //with

   inherited; // call the default Notification code
 end;
```

The above example simply logs the notification events as they come in.

HTML Help training cards

HTML Help Training Cards are similar to the WinHelp variety. The HTML Help provides a training card command that can be embedded in a HTML document, in the same way as you embed HTML Help shortcut commands. The training card command allows you to send a WM_TCARD message to the owner application with user specified wparam and lparam values. The lparam can be a string if you wish.

Example 4 on the sample code CD shows you training cards in action. Open the "Other" menu to run the training card example.

Setting up training cards

The first thing to do is to insert the HTML Help training card command into the document. Like the HH shortcut command, we can choose to insert the command as a visible button or as a stub that is called by script.

1. Open HTML Help Workshop. Select **File › Open** from the main menu and open the document that will send the WM_TCARD message. Position the cursor at the code insertion point. (You can place the insertion point anywhere in the document, though obviously you must take care not to insert code into the middle of another code block.)
2. Click the wizard's hat in the toolbar, or choose Tags › HTML Help Control from the main menu. When the HH ActiveX Control Commands dialog appears, select the Training Card command. *Hhctrl* is always the default prefix, but make sure you enter a unique ID for the control. If the document contains other embedded commands make sure that they all have unique IDs. Click **Next**.
3. Select **Hidden** if you want to call the command using script (e.g. from a hyperlink). Ignore the menus and popup settings. These are a cosmetic bug in workshop 1.2 and have nothing to do with training cards. Click **Next**. Fill in the button information if you selected the button option and click **Next**.
4. Fill in the **WParam** and **LParam** values. The LParam can be a string or an integer. Click **Finished**.

The object code is inserted into your document and looks something like the following:

```
<OBJECT id=tcardtest type=”application/x-oleobject”
  classid=”clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11">
    <PARAM name=”Command” value=”TCard”>
    <PARAM name=”Item1" value=”1001">
    <PARAM name=”Item2" value=”run_example1">
</OBJECT>
```

This won't work. You will need to edit the code so that the two item values are combined into one. This is a known bug with HH 1.2. The code should look like the following.

```
<OBJECT id=tcardtest type=”application/x-oleobject”
  classid=”clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11">
    <PARAM name=”Command” value=”TCard”>
    <PARAM name=”Item1" value=”1001,run_example1">
</OBJECT>
```

Using Script

The above code does not display a visible button so we need to call it using some script. To do this we need to use the HTML Help Click method. Here is an example of a hyperlink calling the object code. Notice we gave the object above an id=tcardtest. If we set id=fred then the code below would need to say “fred.click()”.

```
<a href=”javascript:tcardtest.Click();”>Run Application
  Example</a>
```

The Workshop online help contains some inaccuracies. They mention using the TCard(wparam,lparam) method. Unfortunately this stopped working in HTML Help 1.1b. The only way to call the code via script is using the Click method. This means you need a separate object code block for each training card call you want to make.

Responding to a WM_TCARD

We now have our document with a link that will send a WM_TCARD card message to our application. It's important to note that this won't work unless the application opens the help topic using the application's window handle. Example: HtmlHelp(self.handle, chm, command,0); The HH API only sends the WM_TCARD message to the application that owns the help window.

To receive the WM_TCARD messages, insert the following code into your form class.

```
Uses
  ..., Messages;

type
  TForm1 = class(TForm)
    ...
  private
    procedure WMTCard(var Message: TMessage); message WM_TCARD;
  end;
```

```
 {Help sends WM_TCARD messages here}
 procedure TForm1.WMTCard(var Message: TMessage); //message
WM_TCARD;
 var s: String;
 begin
   case Message.WParam of
     1001:  //Its for us
       begin
         if not IsBadStringPtr(PChar(Message.LParam), maxint) then
           s := PChar(Message.LParam)
         else
           s := '';
         if (s = 'run_example1') then
           Do_Some_Action;
       end;
   end;
   inherited;
 end;
```

In this example the LParam sent with the message is a string. Notice that we have used the window API function IsBadStringPrt() to ensure that we have a string not an integer.

Embedding the WebBrowser control

HTML Help embeds the Internet Explorer reusable control TWebBrowser in each help window so it can display HTML documents. This control is distributed with Internet Explorer as shdocvw.dll. It's easy to embed into your own application. Here's how!

There were a few changes between IE3 and IE4. The IE4/5 shdocvw.dll supplies two controls, TWebBrowser and TWebBrowser_V1. If you are supporting IE3 users or are still using Delphi 3 then you need to use the older version called TWebBrowser_V1. TWebBrowser_V1 has a few less properties, methods and events than the newer TWebBrowser.

Can I install just the shdocvw.dll? MS do not allow you to install just the TWebBrowser control. See the "HTML Help Installation" section of the book for information on installing the Internet Explorer.

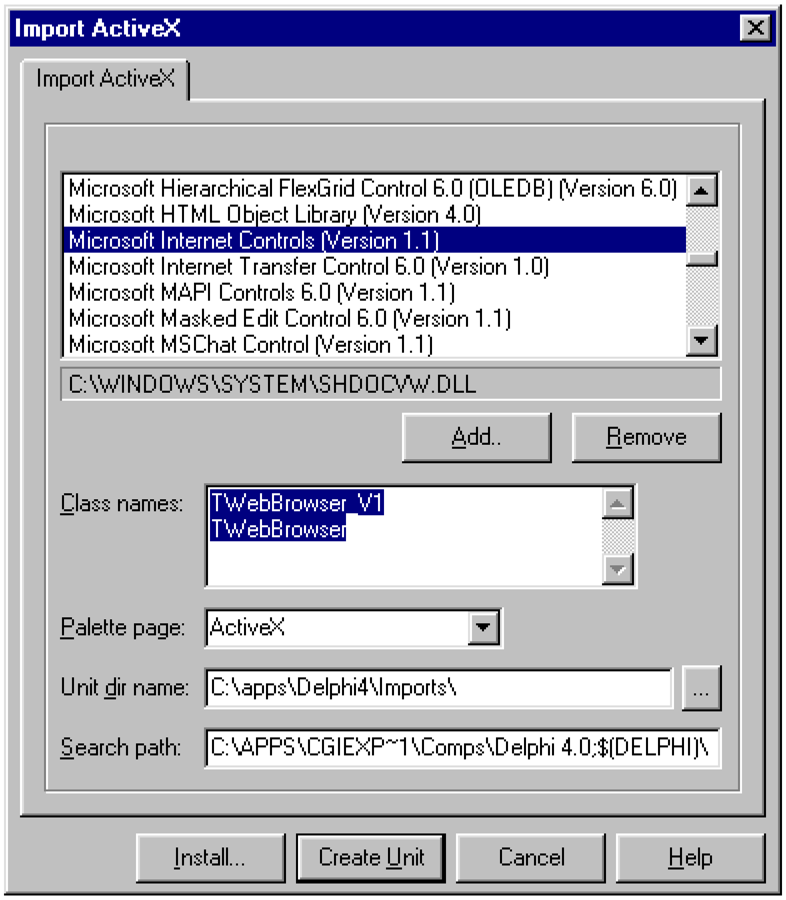
Example 5

You will find Delphi sample code that demonstrates using the TWebBrowser control in the Example 5 folder of the CD. The example comes in the form of a simple web browser so as to best demonstrate all the main browser properties, methods and events.

Importing the WebBrowser ActiveX control

To open the sample project or create an application that uses TWebBrowser, you will first need to install TWebBrowser ActiveX control into your Delphi component palette. Here's how we do it under Delphi 3 & 4.

1. From the Delphi IDE main menu, select **Component › Import ActiveX Control**.
2. When the Import ActiveX dialog appears, select "Microsoft Internet Controls". It has DLL name **%winsysdir%\SHDOCVW.DLL**. Accept the defaults, and click the **Install** button.



Microsoft Hierarchical FlexGrid Control 6.0 (OLEDB) (Version 6.0)
Microsoft HTML Object Library (Version 4.0)
Microsoft Internet Controls (Version 1.1)
Microsoft Internet Transfer Control 6.0 (Version 1.0)
Microsoft MAPI Controls 6.0 (Version 1.1)
Microsoft Masked Edit Control 6.0 (Version 1.1)
Microsoft MSChat Control (Version 1.1)

C:\WINDOWS\SYSTEM\SHDOCVW.DLL

Class names: TWebBrowser_V1, TWebBrowser

Palette page: ActiveX

Unit dir name: C:\apps\Delphi4\Imports\

Search path: C:\APPS\CGIEXP~1\Comps\Delphi 4.0;$(DELPHI)\

Figure 4. Delphi Import ActiveX dialog. Select the Shdocvw.dll.

Notice the two class names. When you click install, two components will appear in the ActiveX tab of the Delphi component palette, WebBrowser_V1 and WebBrowser. Use the older WebBrowser_V1 if you use Delphi 3 or for IE3 support.

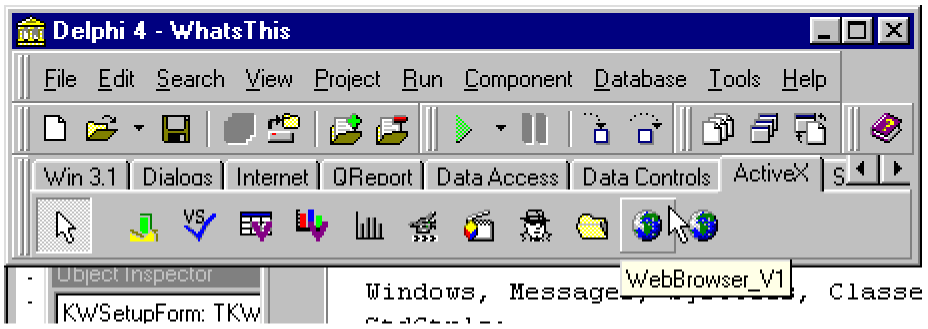

Delphi 4 - WhatsThis
File Edit Search View Project Run Component Database Tools Help
Win 3.1 Dialogs Internet QReport Data Access Data Controls ActiveX S
WebBrowser_V1
Object Inspector
KWSetupForm: TKW
Windows, Messages, , Classe

Figure 5. Delphi ActiveX tab of the components palette.

You are now ready to use the WebBrowser control. As with any of the Delphi visual components you simply drop the component on the form and use it. However there is one word of caution.

Warning: Do not drop the control onto a bare form otherwise resizing could cause access violations. Instead, first drop on TPanel control onto the form and then drop the WebBrowser onto the TPanel control. Even then you may still get a few access violations if you run your application from the Delphi IDE and attempt to resize the window.

Programming the WebBrowser control

Microsoft has provided excellent documentation on WebBrowser properties, methods and events. This can be found either in your MSDN disks if you subscribe to MSDN, or on the public MS MSDN web site.

http://msdn.microsoft.com/workshop/browser/default.asp

http://msdn.microsoft.com/workshop/browser/webbrowser/WebBrowser.asp

Not all properties and methods can be used with WebBrowser control. For example, the property TheaterMode cannot be used. The online help says, “The WebBrowser object ignores the TheaterMode property.” So read the online help carefully.

To open a HTML file in the control, use the Navigate2() method. You can specify local files by using the “file://” prefix. You can open HTML file topics from within compressed help files by using the “ms-its:chmFilePath::TopicPath” type format. In fact you can open any file that you can normally open with Internet Explorer.

The BeforeNavigate2 event occurs just before WebBrowser navigates to a new URL. You also have the option to cancel the pending navigation. This is a useful way to send commands to the application from the WebBrowser control.

For example, the following document link when clicked will fire the BeforeNavigate2 event.

```
<a href="_MY_COMMAND_">Run Application Example</a>
```

In the event code, you would check if the URL parameter contains the text `_MY_COMMAND_.` If it does, cancel the navigation and perform some action. See Example 5 for more.

HTML Help features In uncompressed help

Some HTML Help commands that are embedded into a HTML file only work when compiled to a CHM help file and displayed in a HH window. These include - Alink, Close, KLink, Shortcut, TCard and WinHelp commands. If you need these commands then you should embed an entire HH window into your application. The following commands do work in uncompressed form - Contents, HH Version, Index, Related Topics and the Splash command.

Embedding HTML Help

We can also embed HTML Help windows into Delphi applications.

Example 6

You will find Delphi sample code that demonstrates embedding a HTML Help window in an application in the Example 6 folder of the CD.

In example 6 we have a HH window without navigation tabs that we embed in a Delphi TPanel control. The example has four radio buttons. As each radio button is clicked the HTML Help topic is changed.

How it's done

First the HTML Help window is created using the HH_SET_WIN_TYPE command. Let's quickly run through the code below. Much is just as it was in our example on using the HH_SET_WIN_TYPE command. We need to set fsValidMembers to flag the members we want to set. HHWIN_PARAM_TB_FLAGS is included so we can add the back and print toolbar buttons.

In `fsWinProperties` we have specified HHWIN_PROP_NOTITLEBAR since we don't want the user closing or moving the window. HHWIN_PROP_NODEF_STYLES and HHWIN_PROP_NODEF_EXSTYLES bit flags causes the API to ignore the default window and extended window styles and use only the dwStyles and dwExStyles defined.

HHWIN_PROP_TRI_PANE bit flag is required since we want to see the toolbar pane.

WS_CHILDWINDOW has been included in dwStyles to restrict the window to its owner's window.

```
rcWindowPos := Rect(0, 0, HelpPanel1.ClientWidth,
HelpPanel1.ClientHeight);
```

This sets the HH window to the same size and position as the owner window.

```
fsToolBarFlags := HHWIN_BUTTON_PRINT or HHWIN_BUTTON_BACK;
```

This adds the HH print and back buttons.

```
HtmlHelp(0, nil, HH_SET_WIN_TYPE, DWORD(@wintypedef))
```

This creates a global window type called 'embedwindow'.

```
 Procedure TForm1.ShowEmbeddedHelp;
 var wintypedef: THHWinType; fn, winName: String;
 begin
   winName := 'embedwindow';

   fillChar(wintypedef, sizeof(wintypedef), 0);
   {Fill in the window info}
   with wintypedef do
   begin
     cbStruct := sizeof(wintypedef);  // IN: size of this
structure
     fUniCodeStrings := FALSE;        // IN/OUT: strings not
UNICODE
     pszType := PChar(winName);       // IN/OUT: Name window type
     fsValidMembers :=                // IN: valid members
       HHWIN_PARAM_PROPERTIES    or  // (1<<1) valid
fsWinProperties
       HHWIN_PARAM_STYLES        or  // (1<<2) valid dwStyles
       HHWIN_PARAM_EXSTYLES      or  // (1<<3) valid dwExStyles
       HHWIN_PARAM_RECT          or  // (1<<4) valid rcWindowPos
       HHWIN_PARAM_NAV_WIDTH     or  // (1<<5) valid iNavWidth
       HHWIN_PARAM_SHOWSTATE     or  // (1<<6) valid nShowState
       HHWIN_PARAM_TB_FLAGS      or  // (1<<8) valid
fsToolBarFlags
       HHWIN_PARAM_EXPANSION;        // (1<<9) valid fNotExpanded
     fsWinProperties :=     // IN/OUT: Windows Properties/
attributes
       HHWIN_PROP_NOTITLEBAR     or // (1<<2)  no title bar
       HHWIN_PROP_NODEF_STYLES   or // (1<<3)  only.dwStyles
       HHWIN_PROP_NODEF_EXSTYLES or // (1<<4)  only.dwExStyles
       HHWIN_PROP_TRI_PANE;         // (1<<5)  use a tri-pane
window

     pszCaption := '';       // IN/OUT: Name of a type of window
     dwStyles := WS_VISIBLE or WS_CHILDWINDOW;
     dwExStyles := 0;        // IN/OUT: Extended Window styles
     rcWindowPos := Rect(0, 0, HelpPanel1.ClientWidth,
       HelpPanel1.ClientHeight);       // IN/OUT: window position
```

```
    nShowState := SW_SHOWNOACTIVATE;  // IN: show state
    fsToolBarFlags := HHWIN_BUTTON_PRINT or HHWIN_BUTTON_BACK;
    fNotExpanded := TRUE;             // IN: Nav Bar not expanded
  end; //with

  {** Create the WindowDef ** }
  if Integer(HtmlHelp(0, nil, HH_SET_WIN_TYPE,
    DWORD(@wintypedef))) < 0 then
      ShowMessage('HH_SET_WIN_TYPE failed)
  else
    RdoClick(nil);    {Open the help window}
end;
```

The next part is important. When we open a help topic, we must specify the window type we just defined called "embedwindow". We must also specify the owner as being the TPanel named Helppanel1. Now the rcWindowPos above makes more sense.

Now all that is left to do is create the help window.

```
 fn := 'c:\test\help.chm::/Agent/Genie.htm>embedwindow';
 mHelpWinHandle := HtmlHelp(HelpPanel1.Handle, PChar(fn),
   HH_DISPLAY_TOPIC, 0);
```

The only other segment of code of interest is the TPanel OnResize event. When we open the help window we keep the help window handle returned to us via the HtmlHelp() API call. Using the handle we are able to resize the help window to match the size of the owner window which is the TPanel called HelpPanel1. The SWP_NOMOVE flag has been used so that the x and y coordinates are ignored, since these were set to 0,0 when the help window was created.

```
procedure TForm1.HelpPanel1Resize(Sender: TObject);
begin
  if IsWindow(mHelpWinHandle) then
  begin
    SetWindowPos(mHelpWinHandle, 0,
       0, 0, HelpPanel1.ClientWidth, HelpPanel1.ClientHeight,
       SWP_NOMOVE or SWP_NOACTIVATE or
       SWP_NOZORDER or SWP_SHOWWINDOW)
  end;
end;
```

What About notification messages?

If we want the HH window to send the application HH training cards or notification messages we have a small problem. The owner used in the above call to the HtmlHelp() API function is TPanel.handle. To catch the WM_TCARD or WM_NOTIFY messages sent by the API, we would need to make the owner the form in which case we need to do more work on getting the HH window position right.

Extending HTML Help using COM

The only way to do some help extensions is to write an ActiveX or Automation server. Again I repeat the warning about using Delphi 3 DAX with Internet Explorer. Delphi 3 users should upgrade or expect possible access violations.

If you are having problems with your in-process ActiveX try writing out-of-process server (EXE) instead. Out-of-process servers by their definition run outside the applications address space.

Example 7

The folder named Example 7 contains an example of an out-of-process automation server called helpx.exe.

To register the server, simple run the helpx.exe. Once registered, you should open the example help file help.chm and navigate to the ActiveX Test. Click on the “Click me!” hyper-link and some JScript code is executed that will create an instance of the helpx.helpExtend object and execute a server function.

Note that automation servers work in uncompressed as well as compressed help. Functions in automations servers can display dialogs, return values, set properties, etc.

Calling the automation server

Let’s first look at calling the automation server. Right click the automation test help topic and select View Source. The first thing we see (below) is that script is required. The JScript function first creates an instance of the ActiveX object and uses that object to call one of the server functions. The JScript is called when the user clicks on the hyper link. Note this kind of ActiveX function is only supported by IE4 and upwards.

```
<html>
<head>
<title>Automation Test</title>
</head>
<body bgcolor=”#FFFFFF”>
<script language=”JavaScript”>
function DoTest() {
  var xx = new ActiveXObject(“Helpx.HelpExtend”);
  xx.testmsg(“This message box is displayed by the Automation
   server  helpx.exe”);
  }
</script>

<h1>Automation Server Test</h1>
```

```
<p>Click the link to run an function in our automation server
   example. <a href="javascript:DoTest();">Click me!</a></p>

</body>
</html>
```

The above script can also be written in VBScript using the CreateObject() function—e.g. xx = CreateObject("helpx.HelpExtend").

Creating the automation server

Let's run through the steps involved in creating the automation server helpx.exe.

1. Open Delphi and create a new application. In this case the example application is called **helpx.dpr** with a main form called **unit1.pas**.

   If you wanted to create an in-process server (DLL) instead, then you would select **File > New**, select the ActiveX page tab, then the **ActiveX Library** option.

2. Next we must add the automation server component. From the main menu click **File › New**. Select the ActiveX page tab, select **Automation Object**, and then click **OK**.

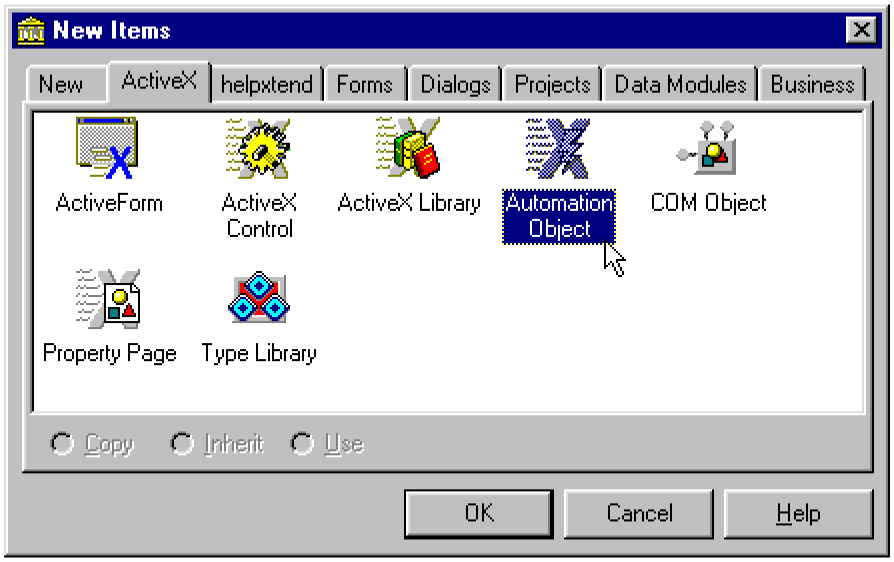
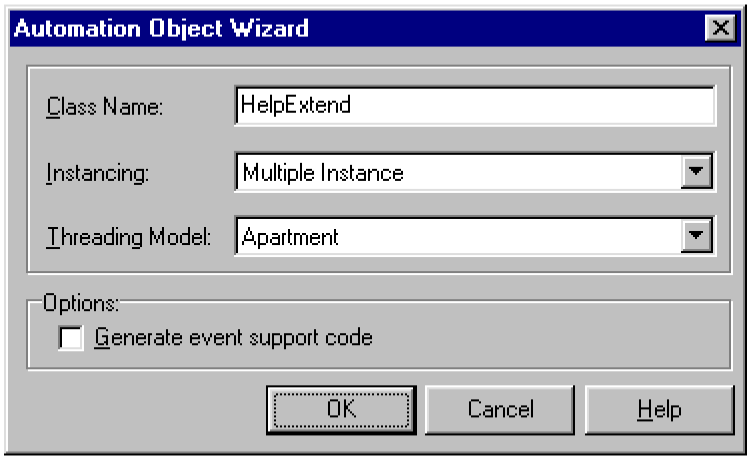


Figure 6. Select Automation Object in the New Items dialog.

3. The Automation Object Wizard dialog will appear next. Enter a class name, select **Instancing** as *Multiple Instance* and **Thread Model** as *Apartment*. The class name combined with the exe name is used to identify the object *exename.classname*. In our example 7, the class name is *helpextend*, which makes the object name used in script *helpx.helpextend*.



Automation Object Wizard

Class Name: HelpExtend

Instancing: Multiple Instance

Threading Model: Apartment

Options:

☐ Generate event support code

Figure 7. The Automation Object Wizard dialog should use the settings shown in this picture.

Click **OK**, and Delphi will create a definition unit for the current project that will contain the code for the Automation object. It also adds a type library to the project. In the test application, the definition unit is called **unit2.pas** and the type library is called **helpx_TLB.pas**.

4. Next we need to create automation server properties and methods. To do this, open **helpx_TLB.pas**, and press F12 to open the Type Library editor. (Using the editor is the only way you can edit the TLB file.) To add a method we simply right-click the IHelpExtend tree item and select **New › Method** from the popup menu. Fill in the method name and use the page tabs on the right to add method parameters and a return value if required. Make sure you use only OLE safe parameters such as WideString, Integer, and UINT. Once you have added a method, the stub will be written to the definition unit, in this case Unit2.pas.

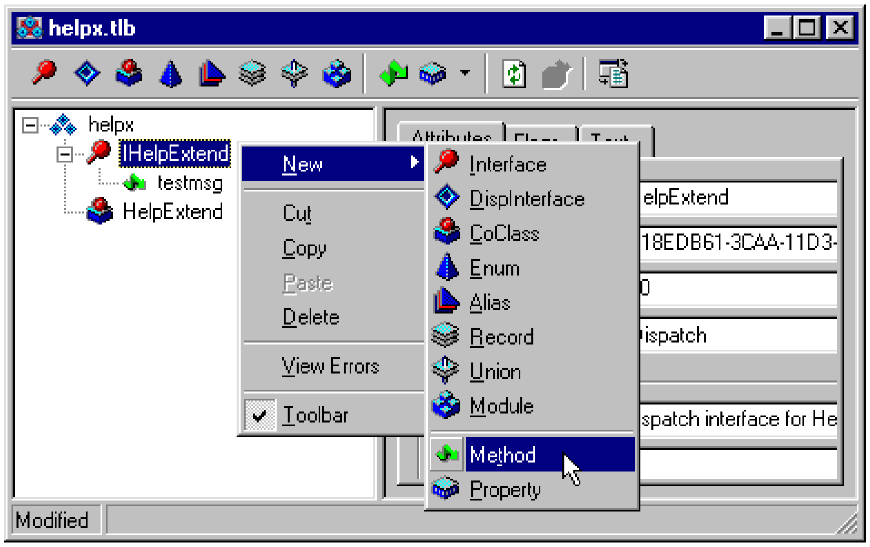


Figure 8. Choose New › Method and set up the method dialog box with the appropriate parameters.

5. Edit **Unit2.pas**, and fill in the stub functions. In the example project there is only one ActiveX function and it simply shows a message box.

Well that's it. I don't want to write a Delphi tutorial. If you want to learn more about Delphi ActiveX you can read the online help or buy yourself one of the many good Delphi textbooks.

Register a COM control as safe

Have you noticed that when you run your COM controls from Internet Explorer, you get those Control may be unsafe type warning messages. This is nothing you can turn off through Internet Explorer options. Search through the Internet Explorer SDK and you will find documentation that will tell you how to setup your COM control to be seen as "safe".

All you need to do is add two registry items. Look at the registry dump below. The first five entries are created when the activeX control is registered. It's the last two entries under the key "Implemented Categories" that tell Internet Explorer that the control is safe. The key CLSID\{GUID}\Implemented Categories\{7DD95801-9882-11CF-9FA9-00AA006C42C4} marks the control as being safe for scripting while the key CLSID\{GUID}\Implemented Categories\{7DD95802-9882-11CF-9FA9-00AA006C42C4} marks the control as being safe to initialize. Also Delphi 3 users will also need to add the line "ThreadingModel"="Apartment" as this was not implemented in Delphi 3.

```
 [HKEY_CLASSES_ROOT\CLSID\{39B0C22C-918C-11D2-96E3-444553540000}]
 @="HelpX Object"

 [HKEY_CLASSES_ROOT\CLSID\{39B0C22C-918C-11D2-96E3-
444553540000}\InprocServer32]
 @="C:\\PROJ\\_OTHER_\\HELPX\\BIN\\MMHELPX.DLL"
 "ThreadingModel"="Apartment"

 [HKEY_CLASSES_ROOT\CLSID\{39B0C22C-918C-11D2-96E3-
444553540000}\ProgID]
 @="mmHelpX.HelpX"

 [HKEY_CLASSES_ROOT\CLSID\{39B0C22C-918C-11D2-96E3-
444553540000}\Version]
 @="1.1"

 [HKEY_CLASSES_ROOT\CLSID\{39B0C22C-918C-11D2-96E3-
444553540000}\TypeLib]
 @="{39B0C227-918C-11D2-96E3-444553540000}"

 [HKEY_CLASSES_ROOT\CLSID\{39B0C22C-918C-11D2-96E3-
444553540000}\Implemented Categories\{7DD95801-9882-11CF-9FA9-
00AA006C42C4}]
 @=""

 [HKEY_CLASSES_ROOT\CLSID\{39B0C22C-918C-11D2-96E3-
444553540000}\Implemented Categories\{7DD95802-9882-11CF-9FA9-
00AA006C42C4}]
 @=""
```

Example 7 has a function in Unit2.pas that, given a class ID, will add these two keys for you.

When Delphi out-of-process automation servers (EXEs) are executed, they automatically self-register. All Delphi out-of-process servers accept the following command-line parameters:

/REGSERVER	Registers the control (default)
/UNREGSERVER	Unregisters the control.

In-process servers (DLLs) must be registered by using another program such as your install program or by running the window program RegSvr32.EXE.

Extending HTML Help using KeyHelp

If you really want to program HTML Help make sure you check out KeyHelp by KeyWorks. With KeyHelp you can create your own CHMs and ITS files, access files within a CHM, create true rich HTML popup windows plus much, much more.

Example 11 shows you how to access the KeyHelp.ocx interfaces from Delphi. You can download the latest copy of KeyHelp, including the full documentation for this product, from:

http://keyworks.net/keyhelp.htm

Reading the contents of a CHM file

Delphi code example 10 shows you how to access the contents of a CHM file. Run the demo and open up any CHM or ITS file. The left side displays all the files found in the CHM while the right side displays the contents of the selected file as either a hex dump or HTML page.

The help compiler creates the first several files in the CHM. These files contain special information such as your window definitions, context help mappings, full text search info and so on. These files all have paths that start with the character “#” or “$”. Following these files are the help topics and image files.

CHM files are stored in IStorage format. Ralph Walden has published a C++ header file CITS.H on his web site that provides access to the IStorage file structures. Example 10 provides a port of this file called HH_ITS.pas.

Novice Delphi programmer may need to brush up on COM to understand the code provided in the example.

A few final remarks

The API commands HH_INITIALIZE, HH_PRETRANSLATEMESSAGE and HH_UNINITIALIZE have not been covered in this chapter. They are supposed make HH windows run in the same thread as the host application. These commands did nothing for me. After many hours of trying I gave up. If you have any luck using these commands please let me know.

I hope this chapter has provided some insight into how you can go about programming HTML Help under Delphi. I will be happy to try and answer questions if you have any. You can email me at HHbook@HelpWare.net. ■

About the author, Robert Chandler

Robert Chander works full-time for Varian Australia, makers of optical spectroscopy instruments. After a decade in software development, Robert moved into Varian's Marcom department to work with help systems, Web, multimedia, and video. Robert spoke at the 4th Annual Australasian Online Documentation conference (April 2001) on behalf of MicroSoft, where he introduced Microsoft Help 2.0. Robert is a Microsoft Help MVP (Most Valued Professional), which explains his association with the Microsoft help development team. Robert's HTML Help shareware program, FAR, is considered a "must have" utility if you are authoring HTML Help or Help 2 without a commercial help authoring tool.

About the publisher, Help Think Press

This white paper was developed under contract to Help Think Press, a division of Work Write, Inc. Help Think Press provides publications of interest to user assistance and performance support specialists. Work Write, Inc. is a consulting firm that specializes in the design and development of user assistance and performance support for the Windows and Web platforms. Work Write, Inc. president Cheryl Lockett Zubak is a charter member of the HTML Help MVP program, and teaches courses for help authors on RoboHELP HTML, ForeHTML, Dreamweaver, Homesite, and help design/architecture. With her partner, Ralph Walden, Cheryl is also developing tools and strategies for enhancing the capabilities of Microsoft HTML Help, particularly for developing embedded help systems and performance upport (EPSS).

Help Think Press
A Division of Work Write, Inc.
128 South Eastview Avenue
Feasterville, PA 19053 USA

Phone: +1 215.357.3453
Fax: +1 215.357.0695
Web: http://www.workwrite.com/helpthink/welcome.htm
Email: helpthink@workwrite.com

Appendix 1

The following table shows the HTML Help commands available.

HTML Help Command	Command Description
HH_CLOSE_ALL HH_GET_WIN_HANDLE HH_GET_WIN_TYPE HH_SET_WIN_TYPE	Close all windows opened by the application. Return the handle of a help window. The last two functions are used to read window type information and to create or modify window types.
HH_DISPLAY_TEXT_POPUP HH_TP_HELP_CONTEXTMENU HH_TP_HELP_WM_HELP	Display text in a in a popup window. The last two commands are identical in behavior. HTML Help 1.x popups can only display plain text at the moment.
HH_DISPLAY_TOPIC HH_HELP_CONTEXT	Display a help topic in help window. Display a help topic by using a Help Context ID.
HH_ALINK_LOOKUP HH_KEYWORD_LOOKUP	Display a help topic using keyword lookup. HTML Help 1.x supports ALinks and KLinks.
HH_DISPLAY_TOC HH_DISPLAY_INDEX HH_DISPLAY_SEARCH	These commands are used to select a tab on the navigation panes of a help window. You cannot select the favorites tab at present.
HH_GET_LAST_ERROR	Return a description string of the last error.
HH_SYNC	Select an entry in the table of contents.
HH_INITIALIZE HH_PRETRANSLATEMESSAGE HH_UNINITIALIZE	These functions are used to run a help window in the same thread as the application. Useful for developers wanting a very tight binding between application and help window. Example: Embedded help systems.

The next table lists the WinHelp commands and how they relate to the HTML Help.

WinHelp Command	HTML Help Equivalent
HELP_COMMAND	There are no help macros in HTML Help. Instead we use Script, Java applets and ActiveX to extend help topics.
HELP_CONTENTS HELP_INDEX HELP_FINDER	In HTML Help the contents and index are built into the help window. The closest commands are HH_DISPLAY_TOC and HH_DISPLAY_INDEX. A HH_FINDER is defined but this simply does the same as the HH_DISPLAY_TOPIC command.
HELP_CONTEXT HELP_CONTEXTPOPUP HELP_CONTEXTMENU HELP_WM_HELP HELP_SETPOPUP_POS	HTML Help uses HH_HELP_CONTEXT to display a help topic in a window but cannot yet display topics in a popup window. HTML Help can display text in popups using HH_DISPLAY_TEXT_POPUP, HH_TP_HELP_CONTEXTMENU or HH_TP_HELP_WM_HELP. Setting the popup position is now built into these commands.

WinHelp Command	HTML Help Equivalent
HELP_FORCEFILE HELP_HELPONHELP HELP_SETINDEX HELP_SETCONTENTS HELP_TCARD	There are no equivalent HTML Help commands. There is a form of TCard available in HTML Help.
HELP_KEY HELP_MULTIKEY HELP_PARTIALKEY	HTML Help can display ALinks and KLinks using HH_ALINK_LOOKUP and HH_KEYWORD_LOOKUP.
HELP_QUIT	HTML Help does not require you to terminate the help session. Although you should call HH_CLOSE_ALL before you shut down the application or risk access violations.
HELP_SETWINPOS	No equivalent command. To position or resize the help window, pass the help window handle to a windows API function.