Live Nude Documentation!

Doug Hennig

Creating documentation, especially user manuals and help files, is usually the last thing a developer wants to do. HTML Help Builder makes creating HTML Help files actually a fun thing to do!

Sorry about the manipulative title, but I figured with the word "documentation" in the title that, being a programmer, you'd simply turn the page and move to the next article <g>. In reality, we're going to discuss a new tool from West Wind Technologies called HTML Help Builder.

If you've built HTML Help (CHM) files using Microsoft's HTML Help Workshop, you know it can be tedious to manage all the files that make up an HTML Help project. I've used other tools, like HDK, to assist with help file creation, but they're frequently add-ons for Microsoft Word that have awkward interfaces and somewhat limited functionality. West Wind's HTML Help Builder makes it easy (and dare I say, fun) to create HTML Help files. It was developed by Rick Strahl, a well-known VFP guru, initially as a tool to use for his own documentation efforts. However, he quickly realized that this would be a useful tool for any developer to use, so he turned it into a commercial product. For details like pricing and availability, see West Wind's Web site (www.west-wind.com).

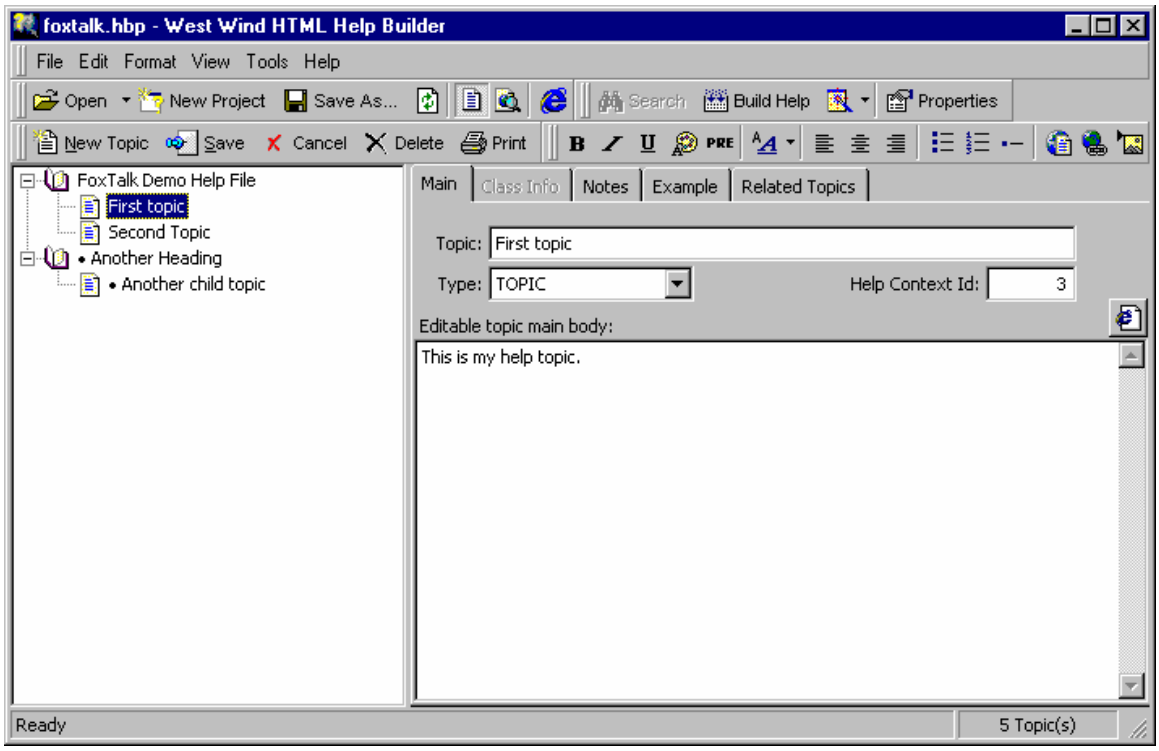
Using HTML Help Builder

I don't have enough space in this article to cover all of the features of HTML Help Builder. Instead, we'll look at it from a "how you'd use it" point-of-view.

The first step is to create a project for your help file. You can either choose New, New Project from the File menu or click on the New Project button in the toolbar. The New Project wizard will appear, in which you enter the title for the project (this is used as the window caption in the HTML Help file that'll ultimately be created), the directory for the project files (it's best to specify a new or existing but empty directory, since HTML Help Builder may create a lot of files), and the filename for the main project file (this file, which is really a VFP table, will have an HBP extension). After the project files have been created, that project is opened and you're ready to go.

Figure 1 shows the HTML Help Builder interface. It consists of a menu bar, several tool bars, a TreeView control showing topics and their hierarchy (initially, there'll only be one topic, which is named the same as the project title), and a display area that either shows a preview of what the selected topic will look like ("Web view") or a pageframe with several pages of editable information about the topic ("edit view"). You can switch between Web and edit view with the appropriate button in the toolbar.

Figure 1. HTML Help Builder's edit view.



To enter information about the selected topic, select edit view. A pageframe with five pages appears beside the TreeView. The Main page shows the topic title, the type, the help context ID, and the main body for the topic. If you change the title, you will be asked if cross-links should be updated, meaning that HTML Help Builder will go through all topics, looking for links to the former name of this topic and change them to the new name. We'll discuss cross-links in more detail later. We'll also discuss topic types in a moment when we create a new topic. Although you can manually edit the help context ID, there isn't a need to since HTML Help Builder will automatically assign the next available help context ID to a new topic.

The editbox for the main body works like any editbox in VFP: you can type or edit text; cut, copy, and paste text; undo changes; and search and replace text. You can also format the text using functions in the format menu or toolbar. Since we're working with HTML, formatting is done with HTML tags. Because you may want to use greater than and less than signs, which are HTML tag delimiters, in your text, HTML Help builder allows you to specify the delimiters it will use for HTML tags; the defaults are << and >>. So, for example, if you select some text and then click on the Bold button in the toolbar, the text will appear as <>Text<>. Other formatting options available are italics, underline, font name and size, color, alignment (left, center, and right), a bulleted list, a numbered list, and HTML styles (such as PRE, H1, and H2). However, you're not limited to these; you can enter your own HTML tags into the text. For example, to create a table, use the <<table>>, <<tr>>, <<td>>, and other table-related tags. This can get pretty tedious, so later we'll look at some tools to extend the functionality of HTML Help Builder. If necessary, you can use HTML directly by placing <!-- RAWHTML --> at the start of the body. This is useful if you're pasting in HTML you created in FrontPage or some other HTML editor.

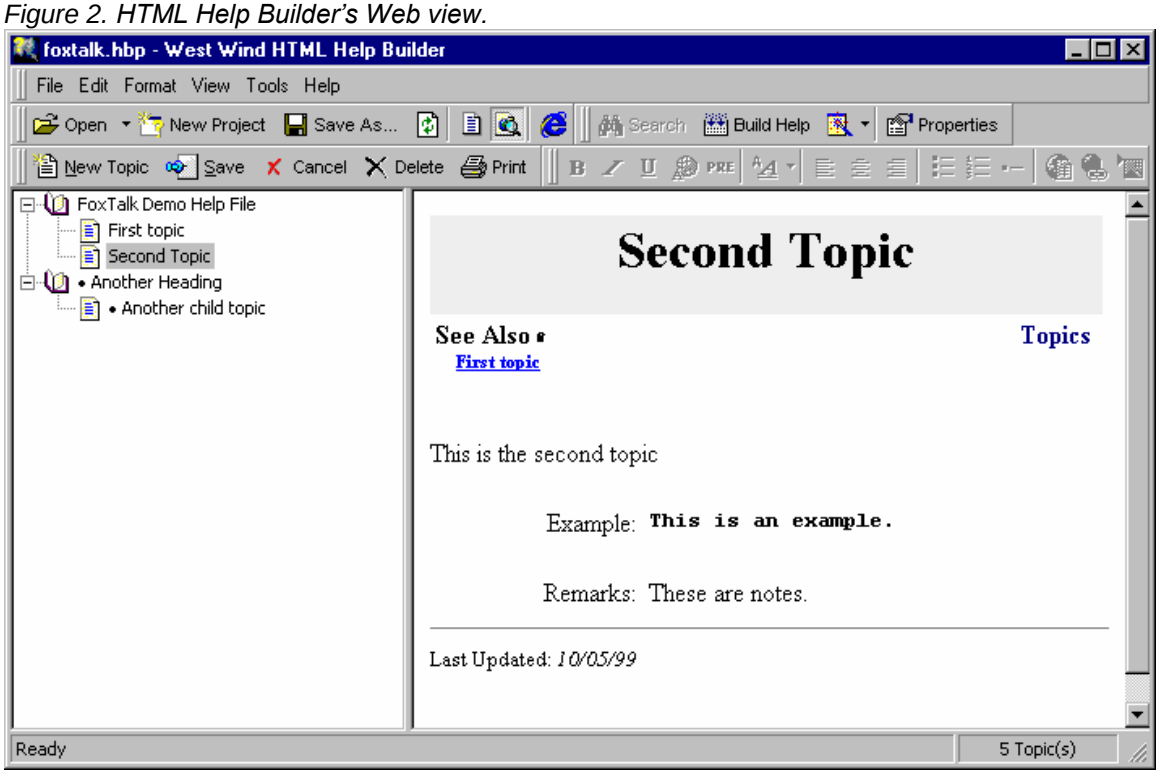
Looking at HTML tags doesn't give you a good idea of what the HTML Help topic will look like, so you'll find yourself switching between edit and Web view a lot. Alternatively, you can tell HTML Help Builder to use the MS DHTML edit control rather than an editbox by clicking on the Browser button above the editbox. This gives you the best of both worlds – an control in which you can edit the content but also see it as it will appear – but according to Rick, it's still a "version 1" control (in other words, it's buggy), so be warned.

One cool thing about HTML Help Builder is its ability to automatically document classes. The Import Class or Type Library function in the Tools menu (you can also choose this function from the Wizards button in the toolbar) allows you to choose one or all classes from a VCX or PRG, or a type library for a

COM object. It creates help topics for each class and the properties and methods of each class, using the comments for each as the body. The Class Info page has additional information specific for classes, such as the scope, syntax, parameters, and return value for methods, and the scope, syntax, and values for properties. Information entered into these items will appear specially formatted in the HTML Help.

The Notes and Examples pages provide editboxes for notes and examples. As with class information, anything entered into these pages will appear specially formatted in the HTML Help; see Figure 2.

Figure 2. HTML Help Builder's Web view.



The Related Topics page allows you to change the parent topic, which you can also do by dragging the topic in the TreeView and dropping it onto the new parent. You can select related topics (which appear under a "See Also" item in the HTML Help; see Figure 2) from a combobox of topics. You can enter keywords for the topic (which can be used to search for topics). You can change the order in which the topic appears under its parent topic, although it's easier to do this by choosing Sort Current Topic Tree from the Tools menu or shortcut menu for the TreeView, and moving topics up or down in the dialog that appears. Finally, you can indicate whether this topic appears as a content topic in the HTML Help file.

Creating Topics

You can create a new topic in several ways: click on the New Topic button in the Toolbar, choose New, New Topic from the File menu, select New Topic from the shortcut menu for the TreeView, select Create a New Topic from the Tools menu, or choose Create a New Topic from the Wizards button in the toolbar (OK, so maybe this is UI overkill <g>). After choosing one of these, the Create New Topic wizard will appear. First, select the type of topic you'd like to create from a combobox of types (we'll discuss types in more detail in a moment). You then choose the location to insert the topic: at the same level as the current topic (which is shown in a combobox and can be changed if desired), at the top level, or as a child of the current topic. Finally, you enter the topic title and choose Finish to create the new topic. The new topic will appear in the TreeView, in alphabetical order (although you can change the order as I mentioned earlier), with a bullet in front of its title, which indicates the topic has no text yet. Once you've entered text for the topic, the bullet will disappear (you may need to click on the Refresh Tree button to make this happen right away, since for performance reasons, the TreeView is only refreshed periodically).

I've mentioned topic types a couple of times, and they're an important concept in HTML Help Builder. The pre-defined types are Topic (used for most topics), Index (contains a link to all other topics, laid out like an index), Header (used for topic group headings), ClassHeader, ClassMethod, and ClassProperty (used for class, method, and property topics). You can also define your own types by bringing up the Help Builder Templates dialog (again, there are several ways to do this, including the View Templates item in the Tools menu) and choosing Add Topic Type.

So, what are types? They're a way to specify the behavior and template for a topic. For example, a Header type appears with a "book" icon in the TreeView while a Topic type appears with a "document" icon. Each type can have its own template, which is a text file containing a mixture of HTML and script code (the default language is VFP, but future releases will support VBScript and JavaScript) that defines what the HTML page created from the topic information should look like. By default, all of the pre-defined types except Index use the same template (WWHELP.WCS in the TEMPLATES subdirectory of your project directory), but you can easily create a custom template for each type if you wish.

Here's the part of WWHELP.WCS that creates a heading for a topic (formatted for easier reading):

```
<TABLE WIDTH=99% BORDER=0 BGCOLOR=#EEEEEE>
<TR>
<TD ALIGN=CENTER>
<h1><%= TRIM(oHelp.oTopic.Topic) %></h1>
</TD>
</TR>
</TABLE>
```

Notice the text between <% and %> delimiters. If you're familiar with programming ASP (Active Server Pages) documents, this syntax is familiar to you: it specifies script to be executed rather than text placed into the output. oHelp is a reference to the HTML Help Builder object, oTopic is a reference to the current topic object, and Topic is the property of the topic object containing the topic title. So, this part of the template creates a table with one row and one column, no border, with a certain background color, and the topic title in the H1 style centered across the page.

As an example of a custom template, I decided that for the "introduction" topic for help files, I didn't want most of the things that might appear in a normal heading or topic, not even a centered title. So, I created a new type, called INTRO, and use this type for the first topic in the help file. INTRO.WCS, the template file for this topic type, looks like this:

```
<html>
<head>
<title><%= trim(oHelp.oTopic.Topic) %></title>
<style>
body { font-family: Verdana; font-size: 8pt;
    font-weight: normal }
</style>
</head>
<body>
<%= oHelp.FormatHTML(oHelp.oTopic.Body) %>
</body>
</html>
```

This template simply sets the title of the HTML page (which doesn't appear in the help topic in the CHM file) to the Topic property of the current topic, defines the font name, size, and style for any text in the body, and uses the FormatHTML method of the help object to render the Body property of the topic (this method is needed to, among other things, convert HTML tags between << and >> delimiters to "real" HTML tags).

In addition to templates, HTML Help Builder supports cascading style sheets. By default, all templates use WWHELP.CSS (in the TEMPLATES directory of your project directory) for style definitions, but you can either edit the LINK REL="stylesheet" tag in the template to point to a different style sheet or edit WWHELP.CSS if you want to use different styles. For example, I decided that all cells in tables should use left horizontal and top vertical alignment, so I added a style for the TD tag that had the formatting I wanted to WWHELP.CSS.

Images, Cross-Links, and External Links

As with Web pages, HTML Help topics can include images, links to other topics in the CHM file, or even links to Web sites. To insert an image into a topic, select edit view, place the cursor where the image should go, click on the Insert Image button in the toolbar (or select Image from the Format or shortcut menus), and select an image from the File Open dialog that appears. In my experience, it works much better if you place the images into the IMAGES subdirectory of the project directory; the HTML Help Compiler seems to have a problem with images in other directories (at least the version I have does).

The ability to link one topic to another is one of the main reasons I like using HTML Help, and is easily done in HTML Help Builder. Select the text you want linked and click on the Insert Bookmark button in the toolbar (or select Insert Bookmark from the Format or shortcut menus) to display the Insert Topic Link dialog. Select the topic to link to from the combobox (if the selected text is similar to a topic title, that title will be selected automatically) and choose OK. You'll find the selected text replaced with something like <<%= TopicLink("selected text","Topic to link to") %>>, which tells the FormatHTML method we saw earlier to insert something like selected text into the output. As I mentioned earlier, if you change the title of a topic, HTML Help Builder asks if you want to adjust all cross-links, since topics are linked by title. Normally, you'll want to choose Yes, but if you decide not to, you can later using the Fix Links function in the Tools menu to go through all topics, looking for cross-links that no longer exist, and providing a means to fix them.

You can create external links, which are usually used to link to a Web site, but can also link to anything the HTML Help engine (which is essentially Internet Explorer wrapped with functionality specific for CHM files) can work with, like Word documents. As with internal links, select the text you want linked and click on the Create Full Hyperlink button in the toolbar (or select External Hyperlink from the Format or shortcut menus) to display the Create HREF Link dialog. Enter the reference for the Web link (including "HTTP://" if it's a link to a Web page) and choose OK. You'll find the selected text replaced with something like selected text.

Creating a Help File

When you're ready to create the CHM file (it doesn't have to be a "final" build; I like to build a CHM file periodically so I can test it), click on the Build Help button in the toolbar (or select the Create HTML Help File function in the Tools menu or Wizard button) to bring up the Generate HTML Help wizard. You have a variety of choices for generation, such as where HTML files should be generated and whether only a subset of topics should be created, but I just usually accept all the defaults (which will create a complete CHM file) and click on Finish. After a few moments of grinding and whirring (well, figuratively), a dialog will appear, displaying the status of the build (including any errors you'll need to correct, such as invalid HTML tags or bad links) and giving you functions to run the help file, run the HTML Help Workshop, view the HTML Help in a browser, or copy the help file to another directory. If everything went well, you should be able to run your help file and see it in action.

Extending HTML Help Builder

HTML Help Builder has a lot of other functionality, including the ability to export subtrees of topics to other HTML Help Builder projects or other applications (like Word). However, hopefully you've been convinced that this is a "must have" tool if you're creating HTML Help files. Now, let's talk about extending the functionality of HTML Help Builder.

HTML Help Builder exposes a COM interface, so you can instantiate it using code like:

```
loHelp = createobject('wwHelp.wwHelp')
```

The CHM file that comes with HTML Help Builder lists the properties and methods of the wwHelp object, so I won't go into a lot of detail here. However, let's look at how this COM object can be used to add new features to HTML Help Builder.

HTML Help Builder includes a program called WWHELP_KEYS. This program adds a wwHelp pad to the VFP system menu, which provides a number of useful functions, such as bringing up the help topic having the same help topic ID as the HelpContextID property of the selected object, and setting the HelpContextID property of the selected object to the help topic ID of the selected help topic. These

functions are useful when you're working in the Form or Class Designer and want to edit the topic for a control or quickly set the context-sensitive help for a control to a particular topic.

Because I frequently work in a non-linear fashion (in other words, I jump around a lot) when creating help files, I often don't immediately finish a topic before moving to another one. Perhaps I need to do some research or want to change the functionality or appearance of the application (you'd be surprised how often you discover you've done something goofy in a user interface when you start describing it in English <g>). In that case, I put a comment preceded with three asterisks (for example, "*** INSERT SCREEN SNAPSHOT HERE WHEN THE UI IS FINALIZED") to indicate that I need to come back and finish the topic. Because there might be a lot of unfinished topics, it'd be handy if there was some way to produce a list of these topics so I can see which ones need to be worked on some more. I started creating a program to do this, but quickly realized that other programs working with the HTML Help Builder would have similar code (such as instantiating the wwHelp object), so I created a wrapper class for wwHelp.

This class, based on SFCustom (our Custom base class in SFCTRLS.VCX), is called SFWWHelpUtils and is located in SFWWHELPUTILS.VCX. The Init method of this class instantiates wwHelp into the oHelp property and loads FOXTOOLS (one of the methods in this class uses functions in FOXTOOLS):

```
with This
  .oHelp = createobject('wwHelp.wwHelp')
  set library to home() + 'FOXTOOLS' additive
endwith
```

After instantiating SFWWHelpUtils, call the OpenHelpFile method to open the specified help file; this method is a wrapper for the wwHelp object's Open method. You might then want to select a specific topic. You can select a topic by its title using the LoadTopicByTitle method:

```
lparameters tcTopic
return This.LoadTopic(tcTopic, .T.)
```

Notice something interesting about this code? It calls This.LoadTopic, passing the specified title and .T. to indicate that the first parameter is a title rather than topic ID. However, if you look at the class, you'll find that LoadTopic doesn't exist. How does this work? The secret is the This_Access method of the class, which has the following code:

```
lparameters tcMember
if not isnull(This.oHelp) and ;
  not pemstatus(This, tcMember, 5)
  return This.oHelp
else
  return This
endif not isnull(This.oHelp) ...
```

This code, which is called on every access to a property or method of the class, passes the access request to the wwHelp object if the accessed member doesn't belong to this class. So, the call to This.LoadTopic in LoadTopicByTitle causes This_Access to be fired, which sees that LoadTopic isn't a method of this class, so it passes the call on to This.oHelp by returning that object from this method. Access and assign methods (like This_Access) make creating wrapper classes easy, because you don't have to manually create the same programmatic interface (that is, create properties and methods of the same name) as the wrapped object; code like the above in This_Access automatically exposes the same interface because any access to properties or methods that don't exist in the wrapper class but do in the wrapped class are passed to the wrapped object.

The GetTopicArray method is a wrapper for wwHelp's GetTopics method, which returns an object containing an array of topic objects. Rather than returning an object, GetTopicArray populates the passed array with the topic ID and the ID of the topic's parent topic, and returns the number of items in the array. This array can then be processed by, for example, calling the LoadTopic method with the topic ID from the array and then doing something with that topic. This method also accepts a filter string so rather than getting all topics, the array only contains those that meet certain criteria.

Here's an example of using GetTopicArray. As I mentioned earlier, I wanted a method to give me a list of all incomplete topics (those with no text or a *** comment). So, I created the GetIncompleteTopics method:

```
lparameters taTopics
return This.GetTopicArray(@taTopics, 'empty(BODY) or ;
  "' + This.cToDoString + '"$BODY')
```

This method expects an array to be passed, and calls the GetTopicArray method with that array and a filter string which requests only topics with no text or the string specified in This.cToDoString (which I've set to ***) in the body. So, a simple routine can call GetIncompleteTopics and display the results in a window or on a report.

How about spell checking? The SpellCheck method in SFWWHelpUtils gets the Body property of the current topic object and uses the spell checker that comes with VFP (specified in the _SPELLCHK system variable) to spell check the topic. Commented out code in this method shows how you'd do it using Word as the spell checker. To spell check all topics, you'd use code like the following:

```
dimension laTopics[1]
lnTopics = loHelp.GetTopicArray(@laTopics)
for lnI = 1 to lnTopics
  loHelp.LoadTopic(laTopics[lnI, 1])
  loHelp.SpellCheck()
next lnI
```

You could even use some logic to avoid spell checking topics that haven't changed since the last time they were spell checked. SPELLCHECK.PRG shows an example of this; it checks the CheckedOut property and if it's empty or less than the Updated property (which contains the last date and time the topic was changed), it spell checks the topic and sets the CheckedOut property to DATETIME().

As I mentioned earlier, it's tedious to enter table-related tags into the HTML Help Builder editbox. So, I created a pair of methods that convert tab-delimited text into an HTML table and vice versa. Text2Table goes through the current topic, looking for tab-delimited lines of text, and adds table-related tags. It uses the cTableProperties property to add additional properties to the TABLE tag (for example, I set this property to "BORDER=1 CELLSPACING=0" to add those properties to the table). The Table2Text method does the opposite: it converts all tables in the current topic to tab-delimited text. This allows you to easily edit it, then convert it back to an HTML table using Text2Table. To see this in action, open the FOXTALK.HBP help project, then run the TEXT2TABLE program. It selects the "Table Topic" topic, which has some tab-delimited text, and creates an HTML table. Switch over to the HTML Help Builder and select this topic, and notice the table tags. Then, select another topic (if you have the topic selected, these methods won't update the topic because it appears that HTML Help Builder locks the record that's currently displayed) and run the TABLE2TEXT program, then switch back to HTML Help Builder and select the "Table Topic" again; you'll see the text was returned to its original state.

In addition to using SFWWHelpUtils (or subclasses of it with additional functionality) or other routines that use the wwHelp object to manipulate the help topics, you can also open the project (HBP) file, which is really a VFP table. The TOPIC field contains the topic title, the BODY field contains the body of the topic, and so on. You can easily write a program to populate or process the topics in this table if desired.

Conclusion

West Wind Technologies' HTML Help Builder is one of those "no brainer" purchases: if you or someone on your development team creates HTML Help files, it just makes sense to use this tool rather than doing things the hard way with HTML Help Workshop. The fact that you can extend its capabilities by working with its COM interface or just by working with the project file directly means you're not limited to what you can do with this tool.

Doug Hennig is a partner with Stonefield Systems Group Inc. in Regina, Saskatchewan, Canada. He is the author or co-author of Stonefield's add-on tools for FoxPro developers, including Stonefield Database Toolkit, Stonefield Query, and Stonefield Reports. He is also the author of "The Visual FoxPro Data Dictionary" in Pinnacle Publishing's "The Pros Talk Visual FoxPro" series. Doug has spoken at the 1997, 1998, and 1999 Microsoft FoxPro

Developers Conferences (DevCon) as well as user groups and regional conferences all over North America. He is a Microsoft Most Valuable Professional (MVP) and Microsoft Certified Professional (MCP). He can be reached at dhennig@stonefield.com.