Building Quality Help Systems

by Robert Palomo

A well designed Help system is a more or less essential component of today's Windows applications. Woe betide the commercial application which is Help-less! If you develop software for internal use, you will still find that the number of technical support calls from irate users will be significantly reduced if you include a well thought-out Help system with your applications.

This article is designed to assist you in the task of building Help systems. Of course, anyone can learn the technical nuts and bolts of assembling a Help system – what is more difficult is to end up with something easy to use and which provides the right information at the right time. So, I've deliberately focussed on the *design* aspects of Help systems here.

Hypertext

Windows Help is fundamentally based on the concept of hypertext. At the risk of over-simplifying, a hypertext system is essentially just a database application. In Windows Help, the data are chunks of text and/or graphics called topics. Each topic is identified by a unique alphanumeric key called a Context String. The Help compiler creates a B-Tree type index of these Context Strings, enabling the WinHelp engine (WINHELP.EXE) to rapidly locate and display any topic in the compiled Help file.

You could have a topic entitled "Using the Widget Control" in a compiled Help file. When the topic was created in the Help source file, let's say it was assigned the Context String `UsingWidget`. When the Help compiler compiles the source files (text and graphics) into binary format, it creates an index of the Context Strings defined in the source files. Within the Help source files, the Help author creates and formats text strings in such a way as to point to a particular Context String. When the user clicks on that text, the Help Engine simply looks up the Context ID in the index and "jumps" to the topic identified by this "key field" value: the Context String. Simple database stuff all the way.

Help Authoring Tools

To create a Windows Help file, you need several tools. At the most basic level, these are:

- A GUI-based word processor capable of saving as Rich Text Format (RTF) files;
- A paint or draw program which is able to save as Windows Bitmap (.BMP) files;
- A screen capture program, if you want to show screen shots in your Help file;
- The Windows Help compiler (HC31.EXE);
- Hotspot Editor for creating hypergraphics (SHED.EXE).

The Windows Help compiler is included with Delphi, but not the Hotspot Editor *[although it **was** included with Borland Pascal 7! Editor]*.

Of course, a dedicated Help authoring package can be used to replace some of these components and the recent ones make Help development very slick indeed. In view of this, there is perhaps little point using the basic technique of hand-crafting an RTF file, so I won't discuss this process here.

Planning For Help

A common pitfall in planning a development project is to either overlook or underestimate the importance of the Help system and the resources that will be needed to develop it. The time and resources required must be planned for in the project. Authoring Help also involves specialized skills that may or may not be available in-house.

You will also want to consider what type of documentation your software requires and how much of it can be furnished on-line. A quick reference or a command reference manual readily lends itself to publication as a Windows Help file. Installation or Getting Started documentation, on the other hand, is needed before the software is installed and should be printed.

Task oriented procedural documentation is a bit of a toss-up and there are different strategies for approaching it. How much of this material should be on-line and to what extent does it duplicate material in the printed Users Guide? There is no single right answer. Are your users adamant about printed documentation? If so, would they accept a Users Guide that covered the basics and then be willing to "graduate" to a high-quality Help system?

Many documentation groups are employing the technique of "single sourcing" their documentation. Single sourcing means that all documentation is written with a common tool and stored in a common format (for example Ventura,. FrameMaker, WinWord, etc). Templates and macros are developed for the tool which can flag text as being print-only, Help-only, or both. A friendly R&D or QA engineer is then recruited to write a utility that extracts the on-line material from the print documentation files and dumps it into Rich Text Format (.RTF) – the format for Windows Help source files. There are some commercial tools to make this easier, such as MasterHelp for Ventura and Doc-To-Help for WinWord.

One component of the Help authoring process that is neglected more often that not is information management. The Help system for even a single software product can involve hundreds of source files (text, graphics, map files, header files, etc). I've seen situations where lots of network disk space was being eaten up with duplicate graphic files simply because there was no quick way to check what

already existed. You should give some thought to tracking components of your Help system early in the design process. A database or spreadsheet can do wonders.

Design Tips

The first screen displayed by a Help file is the Contents topic. Depending upon the scope of the Help system, a single contents screen may or may not be sufficient. In larger systems, the "drill-down" approach may be appropriate. For example, the main contents screen may have a jump for "Menus" which jumps to another contents screen having such choices as "File menu", "Edit menu". In their book *Developing On-line Help for Windows*, authors Boggan, Farkas, and Welinske cite the following design objectives for contents topics:

- Provide a short path to Help information,
- Ensure fast processing of the contents topics,
- Prevent wrong navigation choices,
- Display sibling (related) topics in one glance.

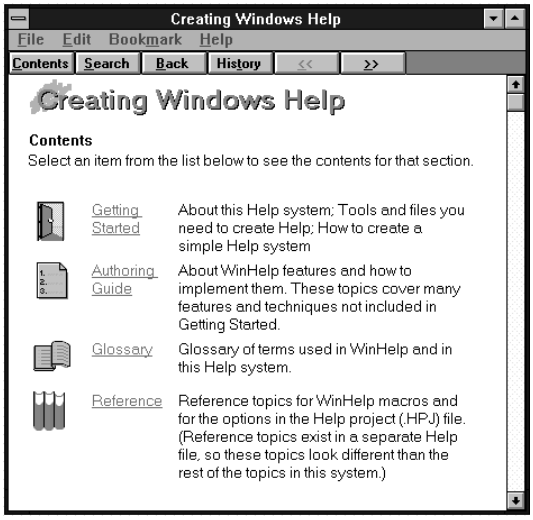
Contents topics are a good place to make judicious use of graphics. Users (including this author) are easily turned off by lines and lines of green text in a Contents topic. Users get their first impression of your Help system from the Contents screen(s), so anything you can do to spruce these up is worth the file size overhead.

The main Help window normally displays a title bar, a menu bar and a button bar. Topic text displays in the regional immediately under the button bar. This area is called the scrolling region. The display scrolls to show any material that won't fit in the area available. In longer topics, this can be a problem because the topic title scrolls out of view. If you provide a "see also" popup near the title, this forces the user to scroll back to the top. You can solve these problems by creating a non scrolling region of the Help window.

You may decide that some information in your Help files is, from a design standpoint, able to stand on its own. In that case, you might consider using a secondary window. For example, in a language reference, code examples for a procedure or function could be displayed this way to enable easy toggle back and forth between the example and the reference topic in the main Help window.

Secondary windows can contain jumps and popups, a non scrolling region and bitmaps just like the main window. They cannot, however, contain a menu or button bar. You can define up to five secondary windows.

➤ *Use of hypergraphics to liven up the Contents topic of the Creating Windows Help file included with Delphi*



Bitmaps And Hypergraphics

Using graphics in a Help file is a matter of trade-offs. On the one hand, Windows is a graphical environment: one picture is worth a thousand words, so why not take advantage of the fact and communicate visually wherever possible? How better to document a menu or dialog box than to show it graphically and enable the user to pop up information about each element? The drawback is, of course, file size. Color graphics significantly swell the size of your Help file. Monochrome graphics are leaner, but not as pretty. You should take into consideration the amount of space available for the Help file when planning the use of graphics.

The Help compiler supports graphics in the following file formats: .BMP, .SHG ("Shed Graphic" bitmaps with "hotspots" for jumps), .WMF, .DIB (Windows device independent bitmap) and .MRB (multiple resolution bitmap).

One pitfall to be aware of when using graphics is that images can appear distorted when the help file is run on a computer with a different video resolution than the computer which was used to create the Help file. If you know the hardware that your end users will be running, you can simply develop your graphics on a computer with the same video resolution.

The Windows SDK *[and Borland Pascal 7 but **not**, apparently, Delphi! Editor]* provides a utility called MRBC (multi-resolution bitmap compiler) that can take separate bitmap files for each video resolution (CGA, EGA, etc) and compile them into a single .MRB file. At runtime, WinHelp checks the hardware and displays the resolution and aspect ratio that most closely matches it. Not many software companies go to these lengths given the decline in CGA and EGA video systems, but you should be aware of the option.

If you are certain that your images will be displayed on VGA systems, you still have the problem of aspect ratio. The best rule of thumb seems to be to create the bitmaps for the lowest common denominator display: 640x480.

One way to get the most mileage out of graphics is to create hypergraphics using the Hotspot editor. This tool imports bitmaps and enables you to define one or more areas as "Hotspots". By assigning the Hotspot the Context String of a topic in the Help file, WinHelp will display that topic at runtime when the user clicks on the image. You

can define a Hotspot as a jump or a popup. Hypergraphics, which have a .SHG extension, are very useful for menu and dialog box help.

Customization

WinHelp provides an array of command macros that enable you to do a wide variety of customizations to your Help system. With these macros you can customize the button and menu bars, access other windows applications, control jumps to specific topics, control the appearance of windows and test for conditions.

With the `RegisterRoutine` macro, your help file can access any Windows DLL, thus limiting customizations only by your ability to write DLLs. For details on these macros search for "macros" in the *Creating Windows Help* (CWH.HLP) file that ships with Delphi. This capability enables you to do some very sophisticated things with the Help system.

Writing For Help

Writing for Help and for print are similar in many ways, but in some ways writing for Help is less complicated. For one thing, you don't have to deal with the issue of how to organize a book. Rather than writing long chapters that flow in a linear fashion, you write a series of relatively short, discrete pieces on a single piece of information.

The trick is to grasp the relationships between these pieces. How will each topic be accessed? What other topics are related to it that a user might want to access after reading this topic? From what other topics might a user want or need to access this topic? Therein lies the challenge for the Help author. I have found that my background in databases has been a valuable asset to me in structuring topic access in Help systems I've worked on because it helps me to perceive these relationships.

When I'm reading a book, I don't mind so much dealing with sentences that are a bit complex. But when I'm in Help, I'm usually there because I'm stuck on something and need some quick info to get me going again. I don't want a treatise, I just want the facts! When I write for Help, I try to think of what's important to me as a Help user and adjust my writing style accordingly. If you're an aspiring Faulkner, I'd discourage you from becoming a Help author. On the other hand, programmers usually tend to be overly terse.

The first rule of technical writing is "know your audience" and this applies as much to Help as to print. You'll write for Novell Network administrators differently than you'll write for accounting clerks.

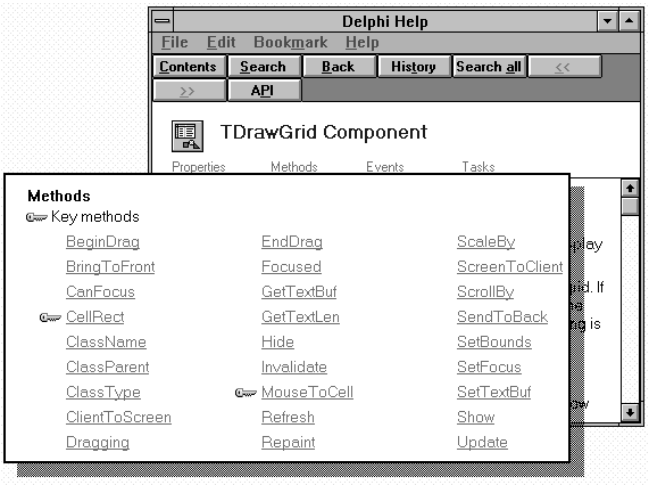
What about converting existing print documentation to on-line Help? This is a thorny issue and there's no single answer. In the case of reference documentation, the conversion is relatively easy, but for task oriented material there are a lot more grey areas. If the user guide is well written in the first place, then the job of chopping it up into Help topics is usually feasible. If the printed manuals are verbose and/or incomplete, you may find it easier to get the manuals into shape first and then hack them up into Help topics.

Further Information

In an article like this, it's only possible to scratch the surface of the subject of Help authoring. If you want to know more, there are a couple of good resources. Check out the Help file *Creating Windows Help*. In a normal installation there's an icon for it in the Delphi program group, otherwise, look for CWH.HLP in the \DELPHI\BIN directory. This provides pretty comprehensive information about the mechanics of creating and compiling a Windows Help file. You'll probably find enough information there to learn how to create a basic Windows Help system for an application.

If you plan to get into Help authoring to any degree at all, I highly recommend *Developing Online Help for Windows* by Scott Boggan, David Farkas and Joe Welinske, published by SAMS Publishing. This book will take you from pure neophyte to consultant-level expertise in Windows Help development. It's well organized and indexed, very readable and covers all the issues. It includes a disk containing WinWord templates and macros for creating Windows Help source files, example Help projects, Help project file templates, and bitmap graphics you can use in your own projects. It also presents a comprehensive review and comparison of third-party Help authoring tools.

Authoring Help requires you to be part writer, part coder, part QA tester and part graphics artist. You'll find it to be an interesting and challenging part of the software development process.



➤ *Use of a popup in the main Delphi help file*

Robert Palomo has been a technical writer in the software industry in Seattle, Washington and Silicon Valley, California for the past five years and worked as a member of the Delphi documentation group at Borland before taking up his current position. Email him at 76201.3177@compuserve.com