



In the January 2000 issue of *Intercom*, I wrapped up an article on HTML-based help design by noting that one of the growing trends in the development of HTML-based help is the need for standards and designs for help on the Web or, as we'll call it in this article, *Web-based help*. In the past year, this need has become significant. Technical communicators are no longer just experimenting with HTML-based help or trying their hand at a project or two, just to see how it works. Rather, they are implementing these systems in full force. Even more, a significant number of technical communicators are faced with providing

online user assistance for Web sites as well as a new breed of software application: the *weblication* (a software application that resides on a Web server).

Who'd have thought it? Just a few years ago we were still wondering whether we would really have to abandon *WinHelp* for HTML-based information. Now, online help for the Web has become part of the everyday life of the technical communicator.

What is Web-Based Help?

Web-based help, as its name implies, is online help created with the same kinds of file formats and technologies that are

used to create Web sites: HTML in its various forms, scripting, style sheets, Java applets, ActiveX controls, and even XML and Active Server Pages (ASP). You can think of Web-based help as a form of what technical communicators have been calling *HTML-based help*.

Web-based help is a special segment of the more general HTML-based help because it is intended to be stored and accessed on a Web server. Commonly, that server hosts a Web site, intranet, or some variant (e.g., an extranet). Web-based help is not intended to be placed on desktop hard drives, CD-ROMs, or other forms of desktop storage devices.

That being said, many aspects of Web-based help are similar to what we have seen and created for the desktop. In fact, just as with the desktop, the term "help" in Web-based help is a stand-in term for the more general *user assistance*—that is, the various forms of assistance that are provided with a software application, hardware item, service, or other product you are documenting. So when I use the term "help," I'm actually referring to a wide variety of types of assistance: procedures, reference information, user interface elements, wizards, and even animated agents. Just as we have learned to provide these various types of assistance for desktop products, we are also learning to provide them for weblications and Web sites.

Requirements of Web-Based Help

Because Web-based help doesn't reside on the user's desktop computer, it's a unique animal, one that is quite apart from the online help you may have been creating for the desktop using help development systems such as *WinHelp*, *HTML Help*, or *JavaHelp*.

Web-based help has particular requirements that have grown out of its use on the Web. Flout these requirements, and expect to be flamed not only by your customers but passersby who have happened upon your site from search engines. Indeed, if you don't meet these requirements, your customers may never even see your online documentation; they may not be able to view it at all.

Let's take a look at some of the most important of these requirements and the kinds of issues they cause for authors of online help/documentation.

Cross-Platform

Part of the original vision of the Web was to create a repository of information and applications that was independent of the technology used to reach it. The idea itself is fairly basic: When a user accesses the Web from any operating system, whether it be Windows or UNIX, that user should be able to view everything on your Web site.

In this view, your Web site (including the user assistance) should not use platform-specific technologies that would limit the user's ability to view pages as intended. For example, to be fully cross-platform, you would not use a technology such as ActiveX, which runs only on 32-bit Windows. As an alternative, help authors often turn to Java applets, which are intended to be cross-platform. (As we'll see, however, even Java applets may cause problems with other Web requirements.)

An exception to this requirement is an intranet or extranet over which your company has complete platform control. For example, if you're creating information for your company's intranet and you know that everyone is running 32-bit Windows, then using an ActiveX control may be perfectly fine, provided the control meets all other standards set by your company.

Cross-Browser

Life for anyone developing Web-based content would be a lot easier if everyone was using the same browser. However, although most browser statistics report that the greatest percentage of users are using some version of *Internet Explorer*, this may not be true of your customer base. As a result, you must create content that is viewable not just by *Internet Explorer*, but also by *Navigator*, *Opera*, and a wide variety of other browsers.

Why would this fact make life difficult? Doesn't the World Wide Web Consortium (W3C), www.w3.org, standardize HTML? Aren't all browsers just displaying HTML according to these standards?

I'm sure by now you know the answer to these questions: Not exactly. The W3C does indeed attempt to standardize HTML, and, cynicism aside, browser companies do attempt to follow the standards set up by the W3C—eventually. However, the W3C process for developing standards doesn't necessarily follow (or better yet, precede) the development path of the companies that produce browsers. While the W3C is deliberating new standards, the browser companies may be off implementing their version of a proposed standard in hopes that they'll be further along with development than their competitors. As a result, if they release a browser version before a proposed standard has been finalized, or if they release it without completely implementing a standard, those of us who develop content for users of this browser must code to outdated, incomplete, or perhaps even wrong implementations of the W3C standard. Add to this issue the need to support a wide variety of browsers and browser versions, and the problem is even more compounded.

A second issue occurs when browser companies develop technologies that are intended to be used by their browser but aren't supported by other browsers. The ActiveX control (a Microsoft technology) is an example. Despite the fact that ActiveX controls are widely used on Windows and in the development of *Internet Explorer*-only Web sites, the platform- and browser-specific nature of these controls prevents Web sites that use these controls from meeting the cross-browser Web ideal. (Of course, if you only need to worry about 32-bit Windows, then you can certainly use this technology "guilt-free.")

For help authors, the problem of browser differentials creates a number of concerns, such as:

- You need to understand which browsers you must support and set a lowest common denominator browser/version for testing—that is, determine the oldest version of each browser that you plan to support. This is very important, since this lowest common denominator will dictate which level of HTML, or set of W3C standards, you plan to support when coding HTML pages. The best

way to go about this is to check the statistics on your Web site to determine which browser/versions your customers are using. (Make sure you revisit these statistics periodically, since they can change.)

- You must test your online information with many browsers/browser versions. Since multiple versions of the same browser typically can't be installed on a single machine, you must have the resources to do this testing (e.g., multiple boot machines, test machines).
- You must determine the extent to which you can implement those aspects of HTML that are interpreted quite differently from browser to browser. Style sheets and dynamic HTML are common examples, since they are often implemented with wide disparities in various browsers. While it's possible to use scripting to detect browsers/versions and subsequently display different information, the fact is that sometimes the way browsers handle these structures is significantly different. As a result, many authors who develop Web-based help often feel they must rely on fairly simple page layouts. While this isn't entirely true, it is true that developing more sophisticated cross-platform layouts requires a great deal of up-front experimentation and necessitates page development from templates that you have tested and retested on various browsers. In other words, *managing cross-browser development is paramount. Don't expect to develop robust pages without thinking through the impact of a design before you begin to implement it.*

Does this mean that Web-based help can't be "sexy"? Not at all, as we'll see later in this article, but it does mean that ideas for invention can't always be derived from HTML-based help systems on the desktop. If you look at Microsoft *HTML Help* systems for example, you'll see extensive use of Dynamic HTML, which is oriented toward *Internet Explorer* (see Figure 1 for an example). (Yes, *Navigator* supports Dynamic HTML, but this support is often quite different—and, according to many experts, less capable—than that of *Internet Explorer*.)

Zero-Client Installation

One of the most often ignored requirements of Web-based help (and Web-based information, in general) is the *zero-client installation*—that is, nothing that the user needs to view the page, except for the browser itself, should have to be installed on the user's machine.

This requirement is rooted in the notion that Web sites and weblications should be stored in one place and opened from that location, so that the “source code” only has to be updated, maintained, and supported on the server. Another important reason for this requirement is that speed of viewing pages is important for Web-based information. By minimizing the amount of information that has to be downloaded, you also reduce the time it takes for the user to open pages. The goal of a zero-client installation is to require only the page (and accompanying graphical information) to be *cached* (stored temporarily) on the user's machine. No software controls, which must be maintained locally or updated, should be installed.

I mentioned that the zero-client installation requirement is often disregarded; I suppose you might say it has become a guideline rather than a rule. One of the reasons this has occurred is because creating “sexy” zero-client installation pages,

particularly when you want them to be cross-browser, is very difficult with HTML alone.

Luckily, the often-used features of “sexy” Web pages can become so commonplace that they are installed as a matter of course, perhaps even as part of the browser installation. An example is Macromedia *Flash*, a software control that supports vector-based animation. Tools like these provide a trade-off. They're readily available and fairly quick to download. Once installed, they also offer faster display of the types of information that are typically slow to download, such as animation. On the other hand, they have to be updated periodically on the user's machine.

A trend to note: Many companies who have flouted the zero-client installation are rethinking this lapse. The customer's need to view pages quickly and the company's need to minimize costs are too compelling.

Security

In a class I taught last year, a student advised me that even my home office computers are vulnerable to hackers, even though I access the Internet through an Internet service provider (ISP). It was advice I took to heart: I now use *ZoneAlarm* (www.zonealarm.com), a

firewall that protects my computers from outside access. Using ZoneAlarm has been an eye-opening experience that has given me renewed respect for the notion that all online information must be secure.

Let's take a look at the types of features many organizations view as security risks, which often include scripting (e.g., JavaScript), Java applets, and ActiveX controls. I'm not about to get into a war with anyone about whether or not these suspicions are true. (For details on what the W3C views as a potential security risk, go to www.w3.org/Security/Faq/wwwsf1.html.) What's important for us to consider are two questions:

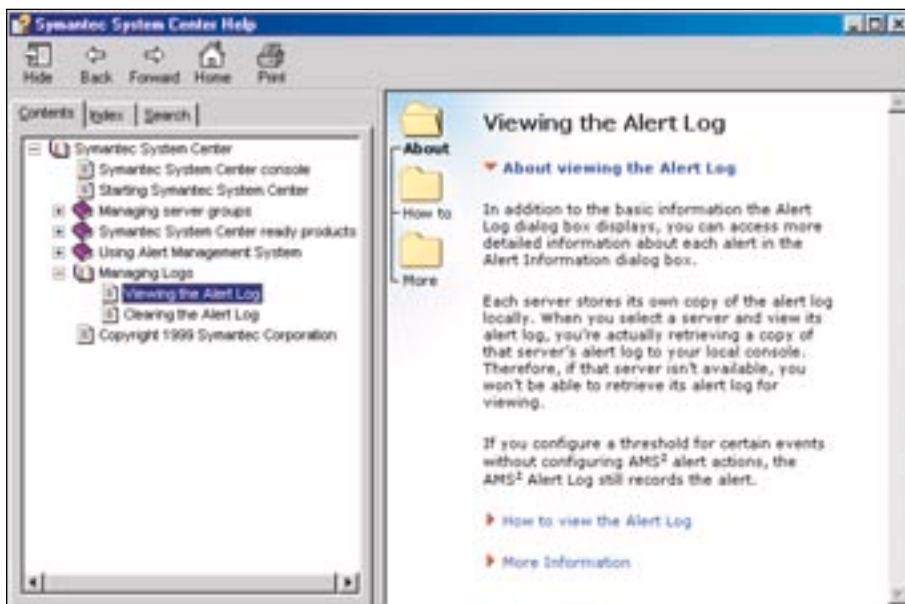
- Does your company or its customers view these features as a security risk? If so, do they ban or restrict the use of these features? If they do, the ability to view pages that contains these features may be switched off in the browser, or pages containing these features may be filtered or blocked through the firewall.
- Does your online information rely on scripting, software controls, or other potentially risky features? What can you do (feasibly or not) to reduce reliance on these features?

As we'll see when we turn to the subject of implementing Web-based help, the security issue is a huge problem for developers of online information, because our authoring tools often build in features that an organization may consider to be security risks.

Implementing HTML-Based Help

The vendors that have traditionally created authoring tools have certainly jumped on the bandwagon to help us to develop online information for the Web. All of the major help authoring tool (HAT) vendors offer some way of creating cross-platform, cross-browser online help. Typically, these tools provide navigational structures that are difficult to create with traditional Web authoring tools, such as a table of contents, index, full-text search, and related topics. (You can see an example of the output from one of these tools in Figure 2.)

Figure 1. Microsoft HTML Help systems, like the Symantec System Center help, often make extensive use of browser-specific Dynamic HTML.



However, these technologies do not come without their potential problems for authors. Here are a few to consider:

- The attempt to provide cross-browser HTML is an enormous effort for tool vendors, and they don't always meet this goal. This means that you must thoroughly understand where your authoring tool might not meet your standards for cross-browser output. Don't be lulled by the promise of "one click authoring." It's a myth. Before beginning to implement your help system, use your authoring tool to develop a prototype of your design and thoroughly test this prototype on multiple browsers. From this prototype, you can develop the templates that I briefly mentioned earlier in this article. *Don't skip this step, no matter how tempting it is to simply launch into content development.*
- In most cases, the navigational structures provided by HATs are created with a combination of scripting (or Dynamic HTML) and Java applets. If your users have determined that these structures are security risks or if your users require a zero client installation, you may not be able to use your HAT to develop your help system. You'll need to consider a more traditional tool. You can still develop nicely designed help with a traditional tool (see Figure 3 as an example), but you'll have to work much harder to create the navigational structures that your HAT provides with a great deal more ease.
- Encouraged by development staffs and Web administrators, some authors are seeking *server-side navigation*. In traditional non-server-side navigation, the cross-platform HAT output produces static pages that contain the table of contents, index, and full-text search. As a result, when you update the help, you often must also update these files. In addition, these static files offer little control for doing things like restricting user access. With *server-side navigation*, a table of contents can be built at the moment the user accesses it, so it is always up to date. In addition, these types of structures can be designed to read the user's "rights" to information

and provide access to only the appropriate pages.

Despite these potential issues, the features a HAT provides can significantly minimize your development time. Sometimes this simple fact outweighs all others.

Design Trends

So far, I've been mostly talking about requirements and strategy. Now let's take a look at a few HTML-based help and online documentation systems that are available on the Web. I've selected them to demonstrate the wide variety of designs for Web-based help.

Integrated with the Desktop

Web-based help is often integrated with its desktop counterparts. Consider, for example, one of the most common architectures of HTML-based online help: a desktop help system that is linked to frequently updated, related information on a Web site. As an example, take a look at Figure 4, which shows the Microsoft Windows Me help system.

This help file is a compiled HTML Help system that is designed to look like Web-based help. The Assisted Support

page provides links to pages on the Microsoft Web site where users can get further information. In fact, the Windows Me help also allows the user to track the status of incidents that have been logged with Microsoft. (I admit that I haven't yet tested this feature to see how well it works. A lot depends upon Microsoft's responsiveness to these logged incidents.)

In this example, you can think of the compiled *HTML Help* system as a kind of HTML-based desktop help, and of the information on the Microsoft Web site as Web-based help. This example demonstrates the trend to layer HTML-based help across physical devices, placing must-have, often-accessed information on the desktop and frequently updated information on the Web.

The Look and Feel of Traditional Help

Most Web-based help systems look and act similarly to the *WinHelp* or desktop HTML-based help documents that you have seen. For example, they provide typical online help navigation: links, table of contents, index, full-text search, and related topics. As I've already mentioned, the HATs make it fairly easy to generate online documentation with this look and

Figure 2. The CompuServe help system, developed with eHelp Corporation's WebHelp, shows the cross-browser output from a help authoring tool.

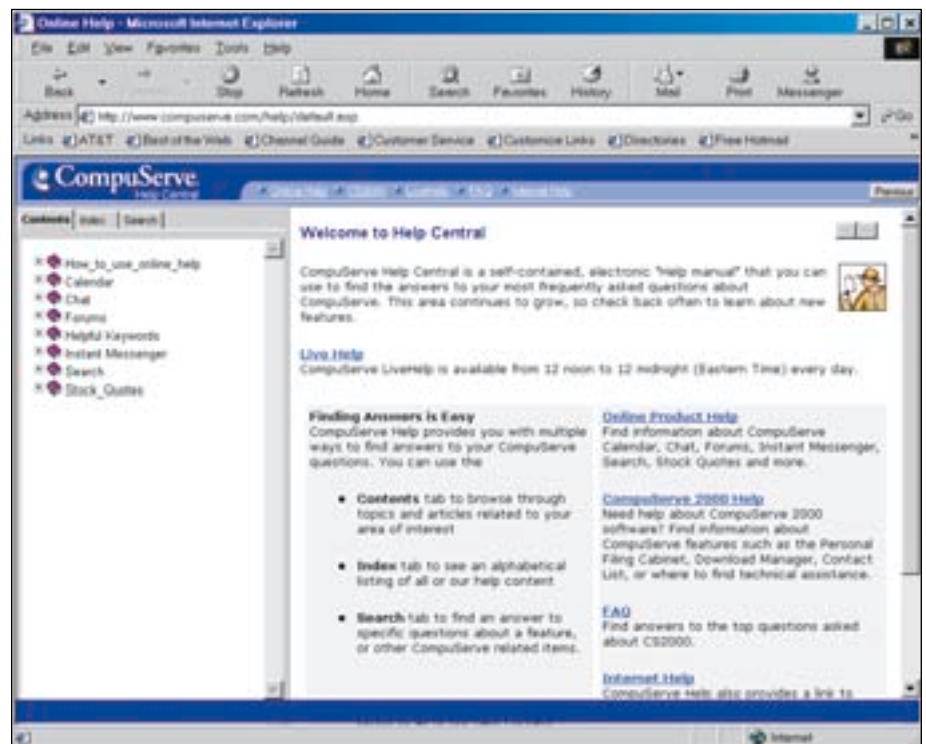


Figure 3. The Big Charts help system shows the type of customized help design that can be developed with a traditional Web editor.



feel. See Figure 2 (www.compuserve.com/help/default.asp) as an example.

Using a HAT *can* result in a somewhat generic look and feel, since these tools are designed to output in somewhat the same way. Compare the more customized design in Figure 3 (www.bigcharts.com) to the HAT output in Figure 2.

Both help systems are cleanly designed and attractive. The second example uses a minimal amount of scripting (potentially reducing security issues) and minimizes the need to download or set particular browser options (e.g., ability to see Java) on the user's machine. Notice, however, that

the "index" in the customized help system is little more than a site map. The HAT-provided index is much more robust.

Access to the Server Provides New Features

The example in Figure 5, which shows the Windows 2000 online documentation, looks similar to the HAT example we just reviewed. For example, it includes a table of contents, index, and full-text search; it also opens in a full-browser view. However, this online document has a feature that depends upon the availability of the server, a feature unique to Web-based help. This feature, called Active Glossary, works as follows: When you highlight a word or phrase from the full text, the Active Glossary immediately locates the closest matching word/phrase in the Microsoft Computer

Dictionary (which is also online). The definition then opens in a small window.

Similarly, the availability of the server makes it easy to display and customize access to *help agents*, those animated cartoonlike creatures that have cropped up in many desktop help systems since Microsoft released Office 97 with its much talked-about Clipit. Agenting technology is actually a mainstay of the Web; for example, popular search engines frequently use them to track Web sites. Even more, however, agenting technology and design simply works better in the server environment, where it is often more possible to closely track and react to the user.

For an example of agents at work in the Web, go to the Agent Ring Web site (www.msagentring.com), which not only uses Microsoft agents to introduce this Web ring, but also links to a wide variety of Web sites that use agents.

Embedded Assistance Is the Way of the Web

It is probably no secret to you that *embedded help*—help that is an integrated aspect of the user interface and behavior of a product—has become a significant trend for desktop help systems over the past couple of years. It has been my

observation that this trend is even more necessary on the Web, where separation of what you're doing from how to do it is antithetical to most Web site design.

An example is field-level help. It's quite possible to find context-sensitive help that acts in much the same way as it does on the desktop: Click some kind of help control and the help for that control opens. The Big Charts Web site (see Figure 3) works in this way. More commonly, however (and in my opinion, more effectively), Web design practices encourage embedded assistance. Look, for example, at Figure 6, which shows field-level help at the Amazon bookstore Web site (www.amazon.com). This field accomplishes the following:

- Provides assistance at the point of using it; it is right on the button you would click to evoke the function of the field (add a book to the shopping cart).
- Assures the user that purchasing is safe.
- Does a little marketing of other Amazon services (the Wish List).

Similarly, procedural information on the Web is also typically embedded. In fact, it is often wizard-like, combining

both software and assistance into the same page. Figure 7 shows another example from the Amazon Web site: a page that opens when you're purchasing a book. Notice the introductory text, which describes what to do, and the form (software) below it. The page also includes a navigational bar (a very wizardlike structure) that tracks how far through the checkout process you've gone.

What's Ahead for 2001?

It's tempting to wrap up this year's article by talking about important online publishing trends, such as XML and single-sourcing/database publishing. However, until the tools and browsers catch up with these trends (and they may this year), I find myself thinking about what I had to say last year about trends for technical communications:

. . . HTML-based help development requires us to form teams of developers—help authors, webmasters, programmers, user interface designers, and artists. Not every team, of course, will include all of these skills. What is clear, however, is that to take full advantage of HTML-based help, you need skills not just in your help authoring tool, but also your help development environment of choice, as well as with HTML tags, scripting, user interface design, and graphic design. We can train for these skills, outsource for these skills, or acquire them through new staff that already have them. But we will need them, for some help systems, to the same degree that Web site development employs them. HTML-based help systems are still new enough that we're trying to get a handle on how these teams will come together.

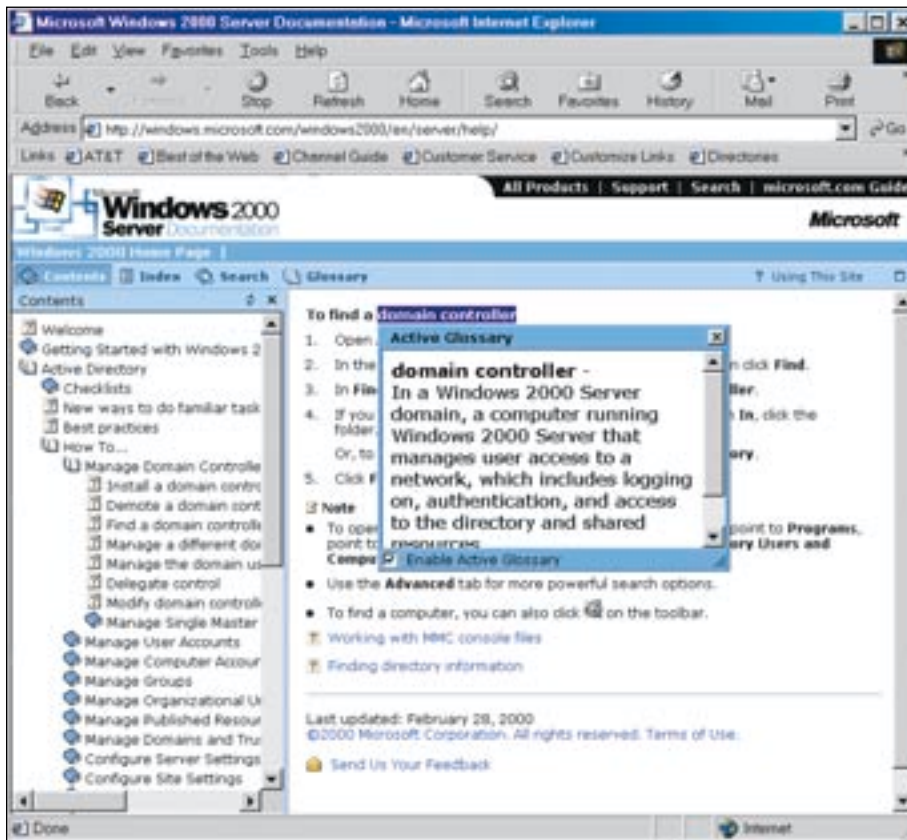
I continue to stand behind this statement, particularly as we do begin to look at technologies like XML and database publishing, which will force us to consider both content and technology in new ways.

Where help for the Web is concerned, these skills are crucial. Without them, it's difficult to understand issues that are basic to the development of Web-based help: How can you design your help system to work in all supported browsers? Is your authoring tool output sufficient? What are the technical requirements of

Figure 4. The desktop help for Windows Me is integrated with information on the Microsoft Web site.



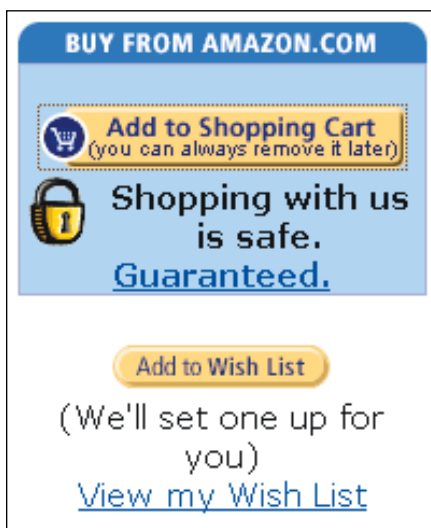
Figure 5. The Active Glossary feature of the Windows 2000 online reference relies on access to the server, a feature unique to Web-based help.



your development environment, and how do they impact your authoring process? How can you go about designing context-sensitive or embedded information?

I also stand behind this statement: These are challenges we are capable

Figure 6. Field-level help for adding a book to the Amazon.com online bookstore is embedded on this button.



not only of managing, but of using to invent new and better strategies for user assistance.

I look forward to looking back next year to see how far we've come. 1

Cheryl Lockett Zubak is president of Work Write, Inc. (www.workwrite.com), a consulting firm that specializes in the design and development of user assistance for the Windows and Web platforms. Cheryl is a charter member of the Microsoft HTML Help MVP program, and was voted "All Time, All Around Help Guru" and "Best Teacher or Instructor" at the Win-Writers 1999 Help Authors Choice Awards. She is coauthor (with Mary Deaton) of *Designing Windows 95 Help: A Guide to Creating Online Documents* and (with Scott Boggan) *Developing Microsoft Help 2.0: A Design and Strategy Guide* (forthcoming 2001). Information from this article is based on a Work Write course on developing HTML-based help. For details on this course, go to: www.workwrite.com/develop_hbh.htm.

Figure 7. The Amazon.com Web site uses a wizardlike design to help you understand the checkout process.

