Browsing Innovations For Books on CD-ROM

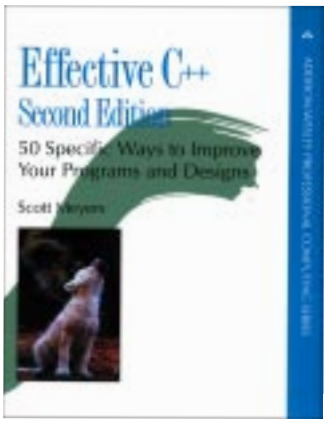
Scott Meyers

The powers of HTML aren't limited to standard Web sites. You can use HTML techniques to make book-based data easier to navigate and read.

This article assumes you're familiar with:

HTML

What should a book on a CD-ROM look like? How should it behave? When I began to think seriously about putting my book, *Effective C++*, onto a CD in the spring of 1998, I didn't know. From what I could tell by talking to producers and consumers of existing CD books, nobody else did either.



One thing I did know was that HTML was my publishing medium of choice. That meant putting a book on a CD boiled down to developing a Web site. I also knew that I disliked the design of most Web sites I'd seen. As you'll see, the design of the *Effective C++ CD* was driven at least as much by what it should not be as by what it should.

My collaborators and I came up with several browsing innovations for the CD. Most ideas are as applicable to Internet-based Web sites as they are to sites published on CD, but some address CD-specific problems. Some innovations grew out of the special demands made by book-length documents, and these innovations, too, are as applicable to traditional Web sites as they are to Web sites masquerading as small plastic discs.

The first innovation I'll examine here is user control over image sizes. Web browsers give users control over the display font size, but they typically give users no control over the size of images. The *Effective C++ CD* allows users to dynamically change image sizes to suit their monitor size, graphics resolution, and personal preferences.

Another innovation is user control over navigation area sizes. Most Web sites have an area reserved for navigational aids, but the size of the material in this area is typically fixed. On the *Effective C++ CD*, users can control the size of the material in the navigation area similarly to the way they control the size of images.

My CD also offers user control over file sizes. How should a long document like a book be broken down into individual HTML files? There are advantages and disadvantages to every solution. The *Effective C++ CD* exploits the fact that CDs are large enough to hold many copies of a book; three copies of the content are provided (each broken down into different file sizes), and users dynamically choose the size they prefer.

Another novelty in the CD is paragraph-specific bookmarking. Most Web sites support the creation of bookmarks at only the top of a page and at major headings within a page. This level of granularity—akin to the index of a book listing only the chapter where a topic is discussed—is inadequate. The *Effective C++ CD* addresses this weakness by allowing any paragraph on the CD to be bookmarked. In addition, each paragraph's unique URL is made easily available, so it's simple to link other HTML documents to any specific paragraph on the CD.

The CD also offers more efficient searching. Most HTML search engines do a poor job of identifying where in a document a particular phrase occurs. On the *Effective C++ CD*, searches identify each paragraph on the CD that satisfies a query.

Finally, the CD preserves link validity. URLs on the Internet are notoriously unstable, but CDs are read-only, hence not updatable. Each link to the Internet on the *Effective C++ CD* goes indirectly through an online table of URLs, so if a URL changes, only the online table needs to be modified; the CD itself remains unchanged.

Each innovation is discussed in more detail throughout this article. To avoid confusion, I should explain that the *Effective C++ CD* actually contains two of my books, *Effective C++* (second edition) and *More Effective C++*.

User Control Over Image Sizes

It has always bothered me that browsers bend over backwards to give you control over how text is displayed, but when it comes to images, you're at the mercy of the Web site designer. Too many Web sites are designed for people browsing at 640×480 resolution, leading to truly indecipherable graphics on my monitor, which happens to run at 1280×1024 resolution. (Soon I hope to switch to 1600×1200 resolution. Life's too short to peer through a keyhole any smaller than it has to be.) Other sites err in the opposite direction, posting enormous images that take a long time to load and, once loaded, dwarf the accompanying text.

One of the design goals for the *Effective C++ CD* was for it

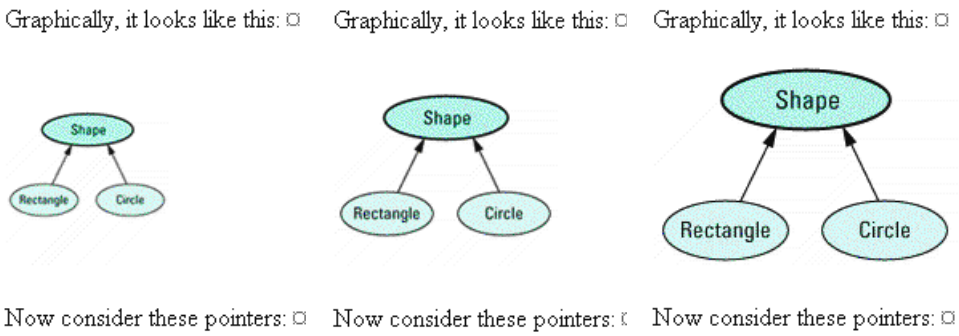
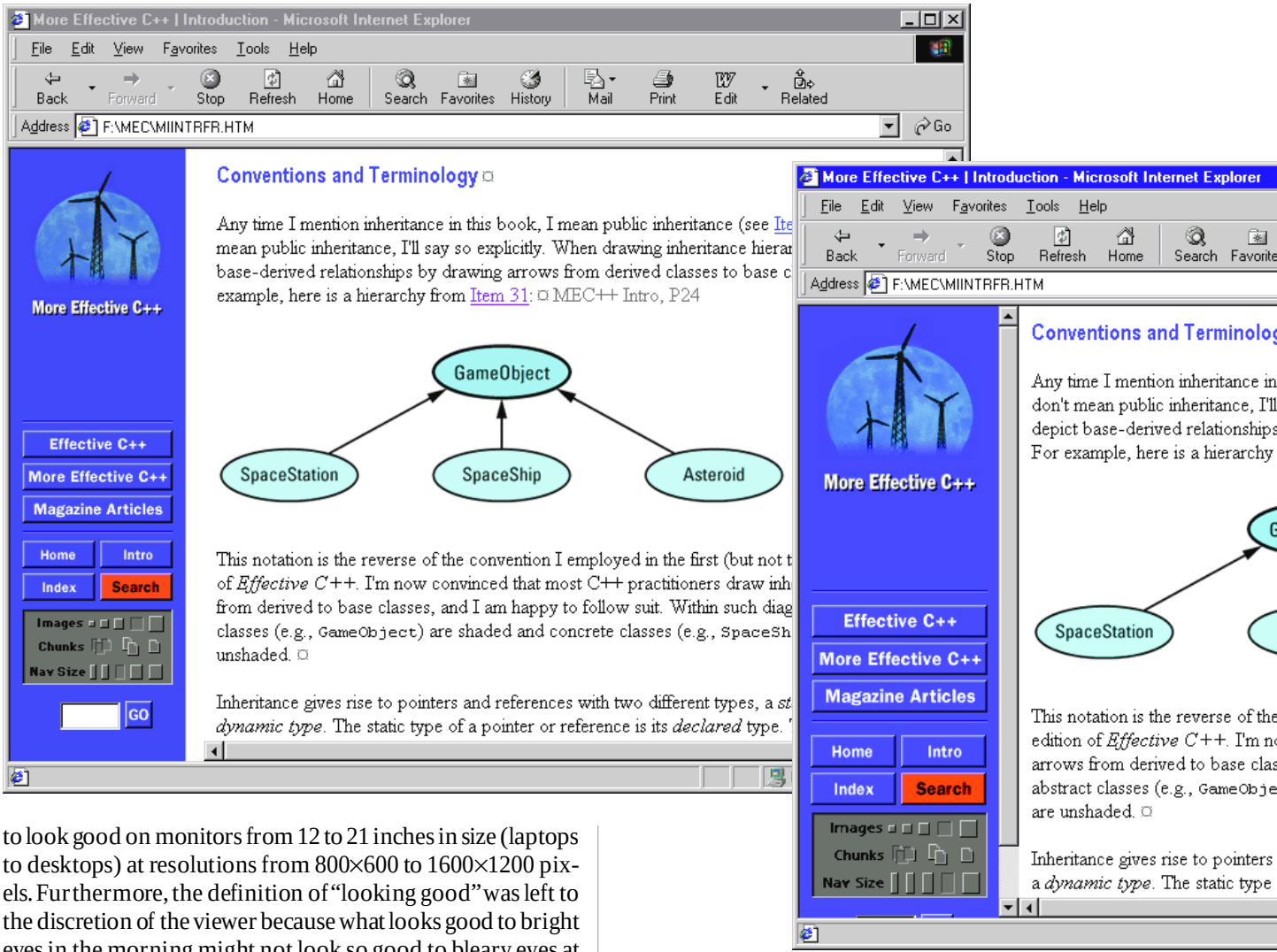


Figure 1: A Selection of Image Sizes



to look good on monitors from 12 to 21 inches in size (laptops to desktops) at resolutions from 800×600 to 1600×1200 pixels. Furthermore, the definition of "looking good" was left to the discretion of the viewer because what looks good to bright eyes in the morning might not look so good to bleary eyes at the end of the day. The ideal solution would be for browsers to take large images and then automatically scale them up or down in some user-defined manner, but browsers don't generally offer this capability.

To deal with this problem, each image on the CD comes in five different sizes, and users can dynamically change their preferred size at any time. Figure 1 shows three of the five sizes (smallest, medium, largest) available for the image from page 172 of the *Effective C++* book (including a bit of text).

When a user requests a different image size, the change occurs immediately; there is no need to reload the document. Thus, it's practical for users to change image sizes depending on the specifics of the image, for example how much detail is present in the image itself.

User Control Over Navigation Area Sizes

As with images, Web site designers often adhere to the one-size-fits-all philosophy for navigation areas. In fact, Web site navigation areas frequently are images. An important difference is that the content of a navigation area is often relatively static. Once you know where the buttons on a navigation bar are located, you might well want to reduce the size of the navigation area to make more browser space available for content. Hence, it's reasonable to want to reduce the size of the navigation area even if you'd prefer to view images within documents at a relatively large size.

The *Effective C++ CD* handles this by offering five different sizes of navigation area. Users can adjust the size of that area independently of the size of images displayed within documents. For example, Figure 2 shows part of the introduction to *More Effective C++*, each showing a different size navigation area (smallest, medium, largest). In each case, the image size is set to the medium setting.

Of course, this is not the only way to give users control over the size of navigational aids. Microsoft HTML Help uses a two-pane design for the client area that gives users control over the location of the split between the navigation and content areas, and Microsoft® Internet Explorer uses a similar design to separate navigation (including searching, favorites, and history) from document content, as shown in Figures 3 and 4.

This is a nice design, but though it allows users to reduce the area allotted for navigational aids, it affords no way for them to scale down the space needed for the information in the navigation area. Instead, the information is simply clipped

on the right and users must grapple with cumbersome horizontal scroll controls (see Figure 5).

The design used on the *Effective C++ CD* allows users to reduce the size of the navigation area while still displaying all of the information within it.

Another approach to this problem is to make the navigation area a window in its own right instead of a pane in the content window. The navigation window can then be minimized independent of the content window, thus maximizing the display area for a document's content while simultaneously making a full-size navigation area available by using a single mouse operation. The Web site for Clarkson University (http://www.clarkson.edu) provides this kind of "remote control" capability (see Figure 6).

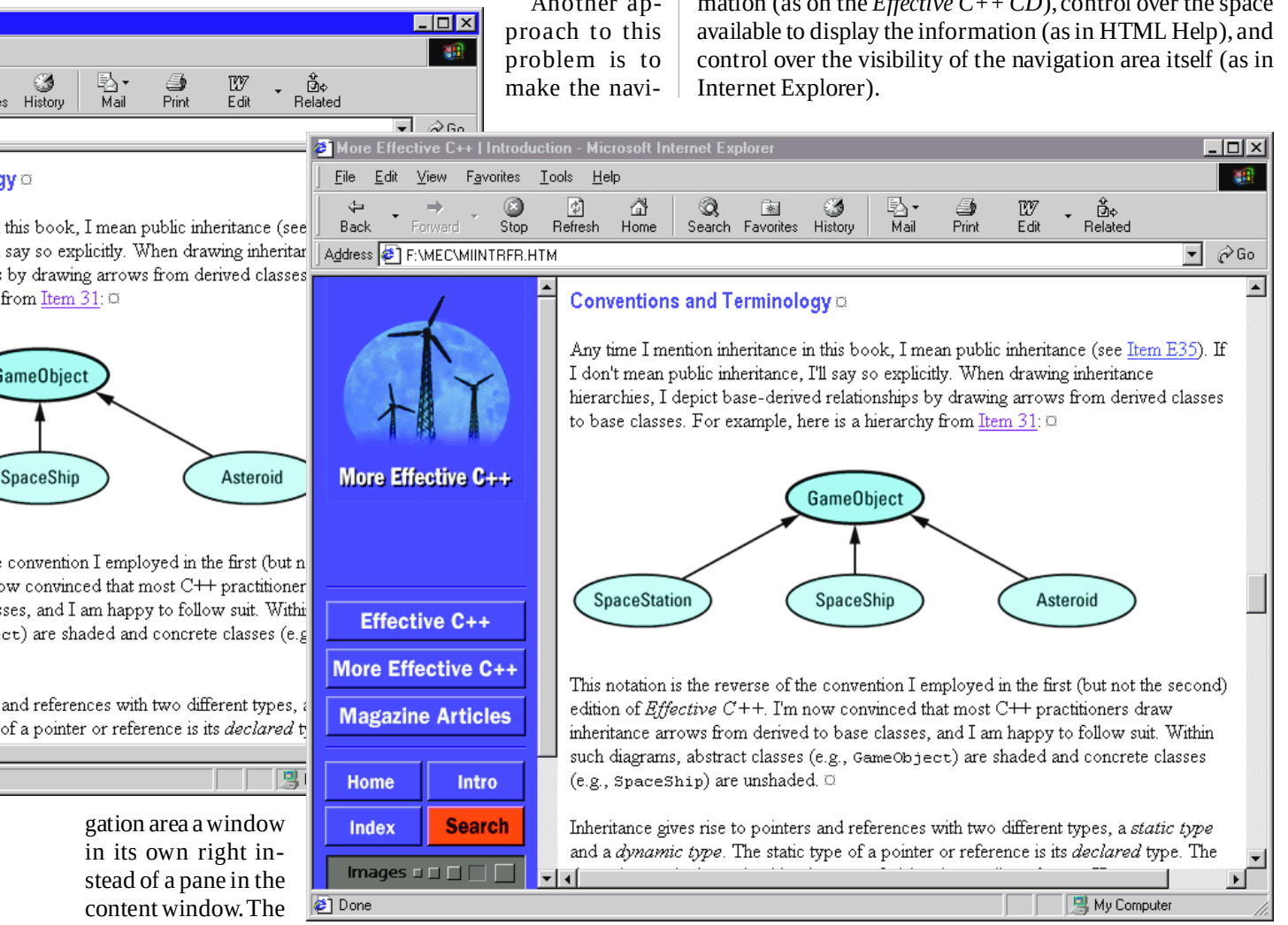


Figure 2: Adjusting the Navigation Area

Internet Explorer gives users the ability to toggle the visibility of the navigation pane. Unlike the design employed by the *Effective C++ CD*, however, neither the Clarkson site design nor the Internet Explorer feature gives users control over the horizontal space required to display the information in the navigation area. (The navigation area in the Clarkson design determines line breaks dynamically, but the size of the text is not adjusted if the window is made narrower. If graphics were used in the Clarkson design, they would presumably be fixed in size.)

An ideal design would probably offer users all three capabilities: control over the space required to display the information (as on the *Effective C++ CD*), control over the space available to display the information (as in HTML Help), and control over the visibility of the navigation area itself (as in Internet Explorer).

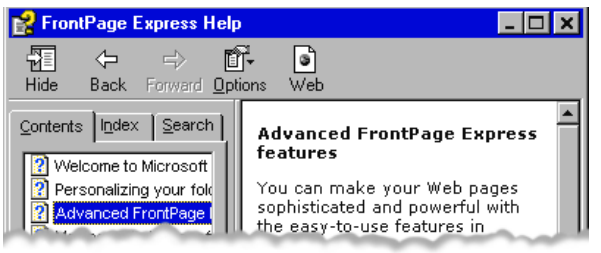


Figure 3: Resizable Panes in HTML Help

User Control Over File Sizes

Everyone agrees that a strength of electronic publication is its support for search capabilities not possible with paper books. Designers of Web sites (including books on CD) generally focus on search engines external to the browser itself because browser searches work on only the current document. I consider this dismissal of browser search capabilities unwarranted. For one thing, browsers already have to parse HTML, so they don't get confused when a user searches for "vector<int>", yet the underlying HTML con-

Virtually all large Web sites are organized in the form of many small files to facilitate bookmarking.

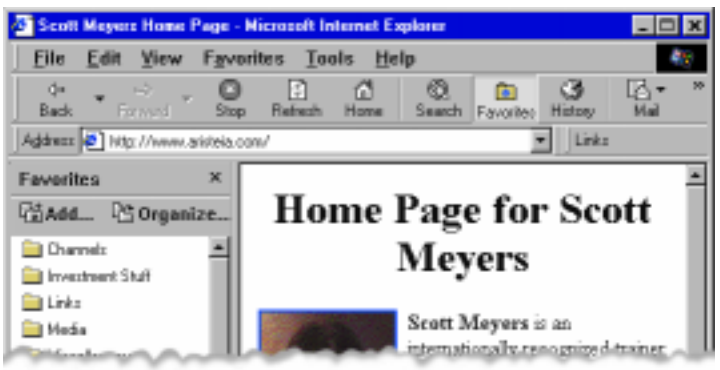


Figure 4: A Two-pane Design in Internet Explorer

tains "vector<int>"; browsers can correctly find a match. More importantly, browsers are uniquely capable of highlighting the text they find, something no external search engine can do. (Some search engines such as deja.com appear to be able to highlight matched text, but they actually generate a new page as a result of each search, then they display that page. This isn't practical for a book on CD, because the CD has no Web server and, at any rate, it hardly makes sense to generate a new copy of an entire book simply to highlight some search hits.)

In the introduction to the *Effective C++ CD*, I summarize the situation—and my way of handling it—like this.

> The big drawback to browser-based searches is that they work on only the current file. It's not practical to put all the information on this CD in a single file, so there's a trade-off to be made. Bigger files support more complete browser-based searches, but the files take longer to load and demand more memory. Smaller files load faster and use less memory, but they yield more limited browser searches. To resolve this dilemma, I decided to let you make the call. This CD contains three copies of each book, each broken down into files of different sizes, and you choose the granularity you like best.

By now, I hope a trend in the design of the CD is emerging. When there are multiple ways to do something and no way is clearly superior to the others, I try to support several or all of them, then let users decide what's best. I've found this to be a much more reasonable way to take advantage of a CD's relatively large capacity than, for example, the inclusion of gratuitous multimedia features such as video clips. More significantly, this philosophy corresponds to my belief that *users* should control their viewing experience—not Web designers. The increasingly large number of magazine-like sites where Web designers control as much as they possibly can—including font choices, graphic sizes, even line breaks—run contrary to the promise of HTML. In an HTML document, the *reader* should be in control. Only the reader knows how much screen space is available for the browser window. Only the reader knows which fonts and font sizes are most readable. Only the reader knows what size graphics are most useful. Designers uncomfortable with this reader-centric view of publishing should eschew HTML and consider PDF instead.

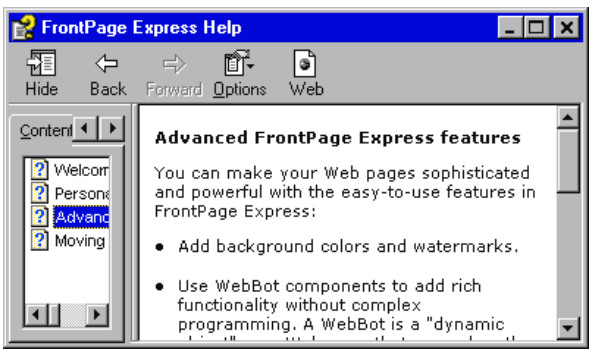


Figure 5: Clipped Text

Paragraph-specific Bookmarking

Virtually all large Web sites are organized in the form of many small files. This is to facilitate bookmarking.

When you set a bookmark using your browser, you simply have the browser remember the most recently loaded URL. That URL certainly corresponds to the file you're viewing, but it need not have a particularly good relationship to your current position within that file. For example, if you've scrolled through the file or used the browser's search function to move around, your current position may be arbitrarily distant from the most recently loaded URL. The farther you are from the most recently loaded URL, the less useful it is to set a bookmark because returning to that bookmark won't put you very close to the information in which you're interested. The traditional solution is small HTML files. That way, bookmarking the file itself will always put you reasonably close to where you really want to be.

As I noted earlier, small files cripple browser-based searches, but for books like mine, small files simply won't work. Some of the discussions in the books span many pages on a single topic. Splitting such discussions into multiple HTML files makes no sense. Furthermore, it's often useful to bookmark material in the middle of such discussions. I put it this way in the CD's introduction.

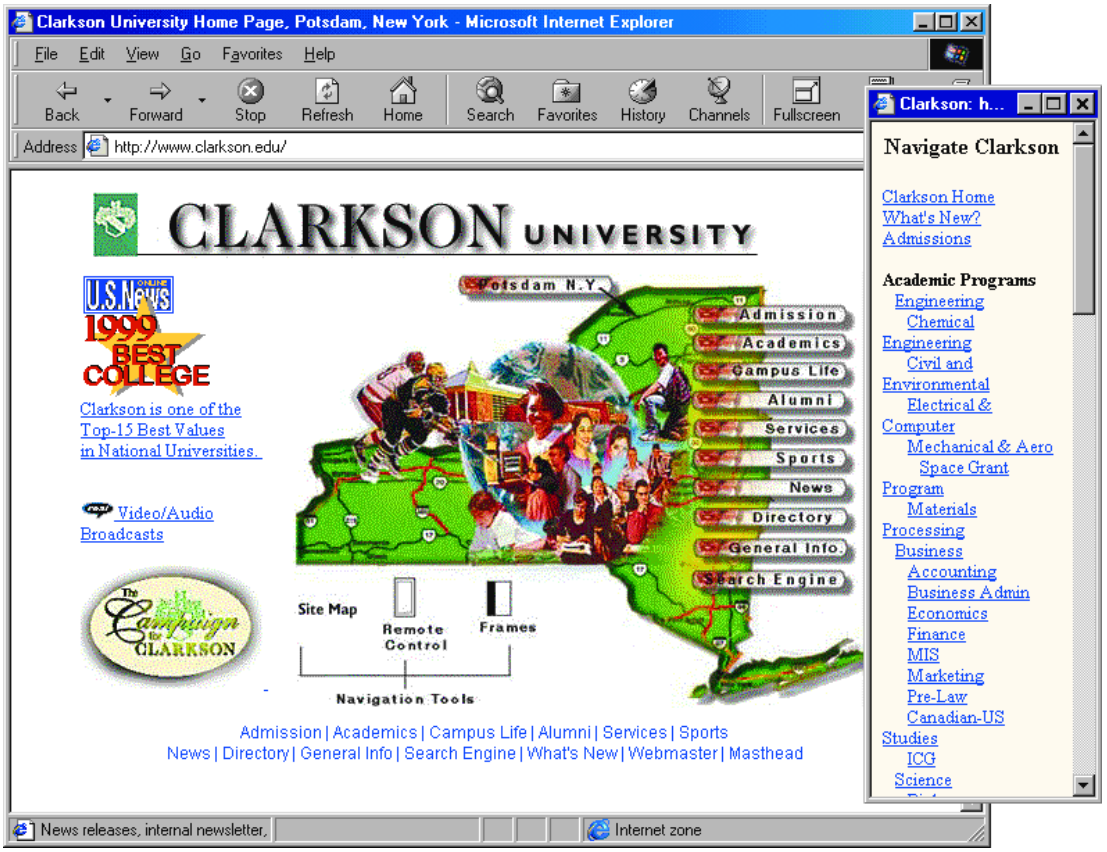


Figure 6: The Navigation Area as a Separate Window

> Item M28 on Smart Pointers, for example, covers more than 20 pages in the printed book, so you might want to create a link that leads specifically to my discussion of how to emulate inheritance-based implicit type conversions among smart pointer types. I address that topic over halfway through the Item, and it's reasonable to want to jump directly to that discussion.

The *Effective C++ CD* deals with this problem in a rather radical way. Each paragraph on the CD has its own URL, and each paragraph on the CD ends with a ¤ symbol that is a link to the beginning of the paragraph. This symbol is called the paragraph's dingbat (see Figure 7).

In theory, it should be possible to bookmark a paragraph by clicking on its dingbat (to jump to the URL corresponding to the beginning of the paragraph), then using your browser's Favorites command in the usual fashion. Some browsers (notably Internet Explorer) fail to implement this functionality correctly, but the CD describes workarounds for such problems. The workarounds are not terribly interesting. What is interesting is that the CD makes it possible to discover the URL for a paragraph by using the dingbat directly.

When the cursor is moved over a dingbat, the dingbat expands into a text description of the paragraph. This is typically an abbreviated version of the document's title plus the number of the paragraph within the document. Figure 8 is the previous example with the dingbat expanded into "CD Intro, P24".

When the dingbat is expanded, browser commands can be used to bookmark it; the text description then becomes the title of the bookmark. Using Netscape Navigator under Windows®, for example, users just right-click the expanded dingbat, then select Add Bookmark from the resulting pop-up menu.

Using similar browser commands, users can copy the URL for a dingbat to the Clipboard. This means it's easy to create links from other HTML documents into the *Effective C++ CD*. That is, not only do paragraph-specific URLs (and the dingbats through which users interact with them) facilitate bookmarking specific information for particular users, they also facilitate the integration of the CD with other collections of HTML documents. For example, I expect many users to link to specific parts of the CD from other HTML documents they have that describe their C++ coding standards.

Because each paragraph on the CD has a number associated with its dingbat, it becomes both possible and easy for people to identify specific locations on the CD. For example, people can refer to paragraph 24 of the CD's introduction or to paragraph 99 of Item 31 of *More Effective C++*. This has proven to be remarkably useful. For example, it enables users of the CD to identify locations of confusion or possible errors in the material and the CD's online errata list can identify locations of defects I have verified. Moreover, the idea of giving each paragraph at a Web site a unique identifier generalizes aston-

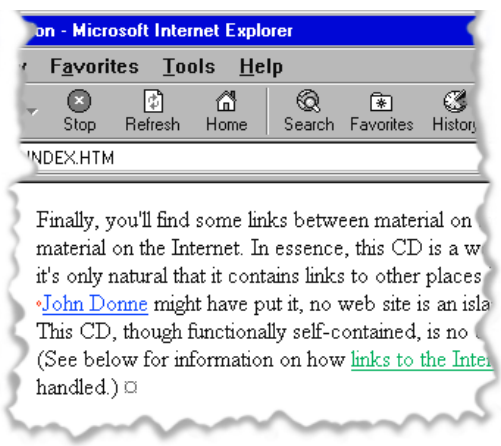


Figure 7: The Paragraph Dingbat

ishingly well; there is otherwise no easy way to specify a particular location within a Web site. Document titles have a granularity that is far too coarse to serve this purpose, yet this is traditionally the only thing Web site designers offer their users.

More Efficient Searching

One of the more annoying things about using search engines (search applications external to browsers) is what I call the double search phenomenon. That's where you have to perform the search twice, once using the search engine to locate the right pages, then again on each page using a browser search to find the correct location on that page. The *Effective C++ CD* avoids this problem because each paragraph on the CD has a unique URL. The CD's full-text search engine locates paragraphs containing the desired text, and each search hit links to one such paragraph. So there is no need to perform a second search. Furthermore, each hit in the search engine is summarized in a context window, so the relevance of a hit can typically be determined entirely within the search applet. In Figure 9, there are 11 hits for the search query "object-oriented programming." At the bottom of the window, the context for the selected hit (paragraph 4 of Item 35 of *Effective C++*—"E35") is shown.

Each paragraph on the CD has its own URL, which avoids the double searches you need to do with many search engines.

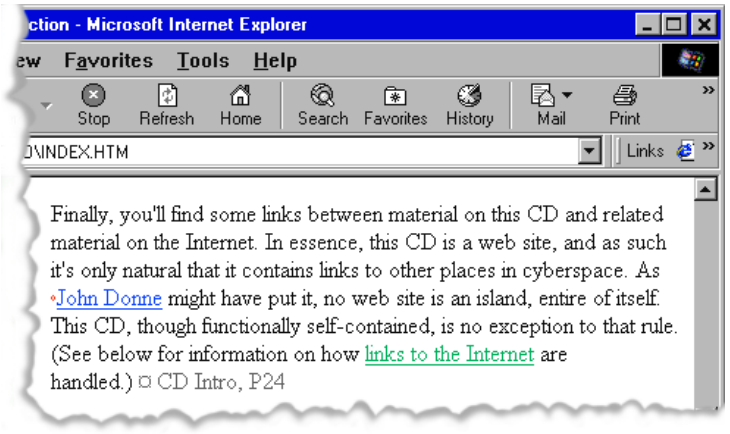


Figure 8: An Expanded Dingbat

Preservation of Link Validity

URLs on the Internet are in a constant state of flux. For reasons unfathomable to me, many site administrators think nothing of reorganizing their sites on a regular basis, rendering all links into the site invalid. This is a serious problem for CD designers because URLs on a CD are, by definition, fixed. It was of special concern to me because I design my publications to have a useful life of at least five years, far longer than one can expect a Web URL to remain unchanged.

In accord with the maxim that all problems in computer science can be solved by an additional level of indirection, the *Effective C++ CD* deals with the problem of unstable URLs by using an online translation table to insulate the URLs on the CD from the actual URLs holding the information to which the CD links. As the CD's Introduction puts it:

> All links from this CD to the Internet go to the Addison-Wesley Web site, where they're translated into the correct URL (to the best of AW's knowledge), and your browser is then automatically forwarded to the correct place in cyberspace. Going indirect via AW's Web site for Internet links imposes a small performance penalty, but I think it's more than made up for by the fact that when a URL changes, all AW has to do is update its translation table, and your CD continues to work.

In practice, I have been pleased to see that the performance penalty is rarely noticeable.

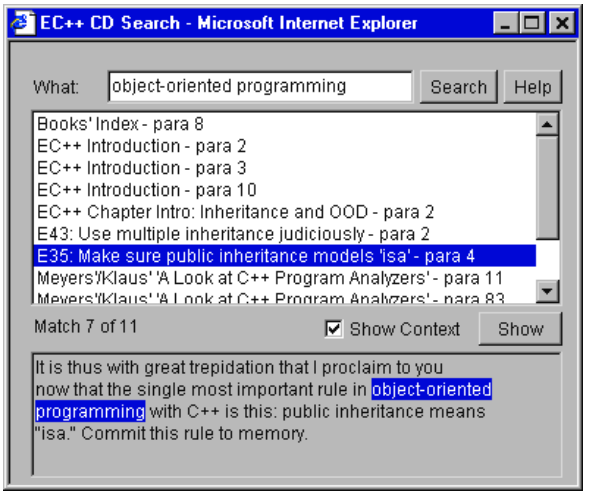


Figure 9: Searching on the CD

Experiencing the CD

Many of the innovations on the *Effective C++ CD* involve dynamic behavior that is difficult to convey in a magazine article. If you are interested in the behavior I've described, I encourage you to visit the *Effective C++ CD* demo site at http://meyerscd.awl.com. The demo is fully functional, and you can experiment with all the features mentioned in this article.