CHAPTER 1

Markup Laid Down

This chapter covers the most basic of basic concepts in any book about displaying information on the Web: What *is* a markup language, and why use one at all? You'll also be introduced to the general characteristics of documents marked up in XML as opposed to other members of its family, and in particular to the ideas that underlie FlixML—this book's customized "language" for describing less-than-blockbuster motion pictures.

If you're familiar with other markup languages, notably HTML, feel free to bound exuberantly over the chapter's first section (Revealing Codes) to the second (The XML Difference), which introduces XML itself.

Revealing Codes

Creating a document via computer was at first no different from creating one with a typewriter: you just pounded away at a keyboard. (Perhaps you began by drafting your words in pencil, then transcribed them into electronic form. I wrote my first book that way, in 1991.)

Even if you're completely new to computers, it should be obvious that the plain old 26 letters of the Roman alphabet (or however many there are in your own language) are inadequate for many purposes. The *meaning* of your words, not just the words themselves, is what is important to your readers. Even if you dress things up with exclamation points and underlining, the most you can hope to communicate in straight, unadorned print is a vague excitement that quickly ceases to hold the reader's attention.

Newspaper and magazine designers have known this for a long time, and augment their plain text with headlines, callouts, and similar devices to emphasize important ideas and to impose a structure to the printed page that would be otherwise lacking.

Shades of meaning

Imagine that you're a late nineteenth-century newspaper publisher. Shortly before election day, you come across a juicy scandal about your biggest competitor, who's running for office. How do you "play" the story?

If you're of the old-fashioned school, you dump the story on the front page with all the other news of the day. It's all set in the same typeface, with only slight variations in size. After all, your words are the important things, right? Responsible readers will read all the news you print, and judge for themselves what's important, right? Well, maybe all that *was* right in earlier times. But with the newspaper industry booming, you know that you've got to do something to catch the eyes of an ever-busier reading public. (It wouldn't hurt if you could stick it to your competitor at the same time.)

So you use all the same words in the story itself. But across the full width of the front page, you shout (with your street-corner newsboys):

Candidate Kane Caught in Love Nest with "Singer"

Ooooh yes. That will get everyone's attention, won't it?

Aside from the size of the type and its placement on the front page, note one other thing about this example (which, by the way, comes from Orson Welles' 1939 classic, *Citizen Kane*[1]): the quotation marks around the last word. A sophisticated reader wouldn't just read the words in the headline; he or she would also catch the added nuance supplied by the punctuation—that the singer in question is really not *much* of a singer.

Those quotation marks (and most other punctuation) are in fact an elementary form of markup. They alter the text, introducing to it a layer of meaning or structure that you can't get from the text alone. Furthermore, in this case, note

1. Decidedly *not* a B movie, although it's got a lot in common with them.

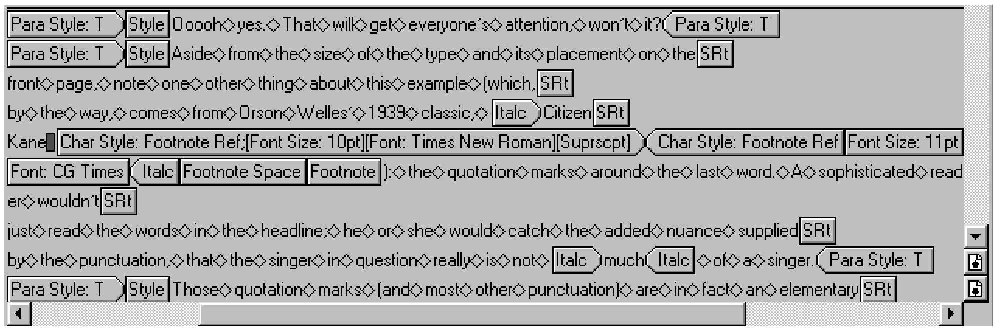
also that the "markup" clearly indicates where the affected text begins and where it ends, and that an opening quotation mark is very similar to, but subtly different from, a closing one. (My sixth-grade teacher called them sixes and nines.)

Markup—whether just in the form of punctuation or on a grander scale, as I'll discuss in a moment—is the simplest way to add layers of meaning to computerized text. But it can do lots of other things, too.

A good illustration of this is the way that the WordPerfect word-processing software keeps track of display styles, fonts, paragraph and document formats, and so on. Yes, now that WYSIWYG displays—fonts, embedded images, all the rest—are universal, WordPerfect does depict a page's visual appearance. But underlying the way the page looks onscreen or on paper, WordPerfect embeds in each document hundreds of bits of information that say, for example, "Begin a left-justified paragraph here, indented a half-inch…beginning here, set the font size to 11 points and italicize it…turn the italicization off here…resume normal font size…end the paragraph here." You can see all these underlying instructions simply by toggling a function key.

Figure 1.1 shows WordPerfect's "reveal codes" window as it depicts a section of the preceding page.

The little pushbutton-like markers within the text can be "edited," after a fashion, by doubleclicking on them. A font marker, for instance, has various attributes (in standard Microsoft Windows terminology, "properties") that can be adjusted through a dialog box that shows a generic sample of how the marked-up text will appear.



```
[Para Style: T][Style]Ooooh◇yes.◇That◇will◇get◇everyone's◇attention,◇won't◇it?[Para Style: T]
[Para Style: T][Style]Aside◇from◇the◇size◇of◇the◇type◇and◇its◇placement◇on◇the[SRt]
front◇page,◇note◇one◇other◇thing◇about◇this◇example◇(which,[SRt]
by◇the◇way,◇comes◇from◇Orson◇Welles'◇1939◇classic,◇[Italc]Citizen[SRt]
Kane█[Char Style: Footnote Ref;[Font Size: 10pt][Font: Times New Roman][Suprscpt]][Char Style: Footnote Ref][Font Size: 11pt
[Font: CG Times][Italc][Footnote Space][Footnote]):◇the◇quotation◇marks◇around◇the◇last◇word.◇A◇sophisticated◇read
er◇wouldn't[SRt]
just◇read◇the◇words◇in◇the◇headline;◇he◇or◇she◇would◇catch◇the◇added◇nuance◇supplied[SRt]
by◇the◇punctuation,◇that◇the◇singer◇in◇question◇really◇is◇not◇[Italc]much[Italc]◇of◇a◇singer.[Para Style: T]
[Para Style: T][Style]Those◇quotation◇marks◇(and◇most◇other◇punctuation)◇are◇in◇fact◇an◇elementary[SRt]
```

Figure 1.1 WordPerfect (version 7) Reveal Codes window

Simplify, simplify, simplify

WordPerfect and similar word processors provide good examples of markup in general, and how it can be used to fancy-up a document so much that—if you're so inclined—the true meaning of the words themselves can be nearly obscured by the way in which the words are displayed. But there are a few problems with these programs—problems that became especially obvious when the Internet started to take off.

First—and perhaps least obvious—there's a problem for the developers of the specific program (like a word processor). A given piece of code in programming language X will almost never run, unchanged, under a different operating system (sometimes even on simply a different computer) than the one on which it was developed. Different computers differ not only in their underlying hardware, but also in their user interfaces. A fully responsible software vendor therefore must commit to developing a version for the Macintosh, as well as a version for the Apple II, a separate version for older MS-DOS computers, one for Windows 3.x computers, one for Windows 95, one for Windows NT, one for Windows 98, one for generic UNIX, one for Sun's SunOS, one for the Amiga, and so on. It's a nightmare, and at the very least, drives up the cost of *purchasing* software for everyone.

Second, there's a problem for users of these programs' output: Typical word-processing files can be significantly bigger than their contents would appear to warrant. The formatting and other style instructions embedded in such a file, even if invisible to a user of the software, of course take up space of their own. These instructions are in so-called binary form—machine-readable, not human-intelligible. It's possible (and common) to use compression software to reduce the sizes of files needing to be shipped around the Internet, but the specific problem in the case of binary files is that they simply don't compress very well. Depending on its contents, a straight text file might be squeezed to ten percent of its original size; the same contents in a word-processing file format, to no less than forty percent or so.

A final significant problem is for developers of *content*: writers, journalists, corporate public relations staff, and so on. It's one thing to prepare a company newsletter, for example, using Microsoft Word, then e-mail a copy to all the company's employees; after all, the information systems (IS) department requires all company-bought computers to have a copy of Word installed. But what if you want to distribute, say, an electronic sales brochure on the Web? Will all your customers have the same word processor, let alone the same *version* of the same word processor? Can you afford to write off the potential customers who don't?

Ch01.fm Page 7 Tuesday, July 18, 2000 10:39 AM

A possible solution for everyone is just to use plain old text files. That's not a very satisfactory answer, though—it puts us right back to the invention of the printing press.

The technology underlying Web-based documents popularized a wonderfully simple solution all around: include markup with plain text content (just like WordPerfect's Reveal Codes mode), and *put the markup in plain text as well.*

If you think about it, there's one potential danger in this solution, however. Consider this bit of a document marked up in a hypothetical language:

```
BOLDITALICSNowENDITALICSENDBOLD is the time for all. . .
```

In this hypothetical language—let's call it ACML, for All-Caps Markup Language—obviously, the markup is simply expressed in all capital letters. When the ACML browser hits such a string of characters, it knows it must treat the enclosed text differently from "normal" text (and, incidentally, not display the markup with the text). But how do you include content that is itself in all uppercase? If you're writing a story in which a character says, "I cannot BELIEVE you'd be so BOLD. Didn't your mother teach you any manners?" how do you keep the browser from boldfacing the entire second sentence, let alone from choking on the BELIEVE because it doesn't know how to display "believed" text?

There's a simple—though not 100% foolproof—answer to this riddle: You design your markup language so that the presence of markup is signaled by special characters that are very unlikely ever to appear in real content.

The first widely successful, nonproprietary markup language, Standardized General Markup Language (SGML), followed this approach. SGML *tags* (as the markup devices are called) begin and end with angle brackets, the < and > characters respectively. This SGML convention has been carried forward in both of SGML's two popular offspring, HyperText Markup Language (HTML) and now the Extensible Markup Language (XML).

Tags vs. what they tag

The discussion that immediately follows this sidebar speaks specifically of *tags.* In common parlance, you will likely hear the term "tag" used interchangeably with another: "element." We'll see much more about the latter term, throughout *Just XML,* but now might be a good time to give you a quick picture of the differences.

Strictly speaking, a tag is not an element. A *tag* is a physical, typographic "marker" that indicates the presence of an element. An *element* is everything from the opening angle bracket of a start-tag to the closing angle bracket of the end-tag (which might be the same physical tag as the start, if the element is empty). An element may contain other elements (delimited by their own tags), which may contain other elements (and so on), and may also contain text content and other forms of markup.

If you really want to get precise, you might want to be aware of two other terms. An *element type* is the overall class to which a particular *element* belongs; that is, an element is a specific occurrence, in a specific document, of its element type. For example, an XML-based language might include a `paragraph` *element type* that may be used in documents based on that language, whereas a given document may contain one or many `paragraph` *elements*.

The other term—not very widely used, but you may encounter it, usually undefined, on XML mailing lists—is *GI*. This is an abbreviation for *generic identifier*; an element's GI is just its name, the "word" that follows the opening < in the tag.

Strung together, these four terms make up a general-to-specific continuum: element type, element, tag, and GI.

Throughout *Just XML*, I'll use the term "tag" when referring to the actual markup device. There's a formal difference between "element" and "element type" that is important to preserve in some cases, and I'll note those cases when covering them; however, by far most usage on the Internet, even among people who know better, is to use "element" for both. That's the convention I'll follow. I don't expect you to come across "GI" again in these pages.

The rules of the markup game

Here's an HTML version of the above ACML passage:

```
<B><i>Now</I></b> is the time for all. . .
```

Notice the features of the tags in this example:

1. Capitalization is not important at all. This was true of HTML, *but is completely changed in XML*. In XML, if there's a `<boldface>` tag, a reference to a `<BOLDface>` tag won't be recognized as the same thing at all.

2. In this case, there are both *start-tags* and *end-tags*, such as the `<B>` and `</b>`. The only difference between them is that the end tag includes an extra character, a slash (`/`). Again, a difference: Not all HTML tags require an end-tag, while in XML all “normal” tags always come in pairs. (Special allowances are made, as we’ll see, for tags that don’t need to enclose anything. A good example from HTML is the `<img>` tag for specifying a graphic that’s to be inserted inline—all the information is within the tag itself. XML has special provisions for dealing with such so-called empty tags.)
3. The separate tag pairs are *nested*. That is, the “italicize this” markup is fully enclosed within the “bold this” markup. (Actually, that’s not always required of HTML documents; browsers can cover for many of a Web-page developer’s human failings, such as improper nesting of tags.) In this example, the word “Now” could be just as well specified with italics first, then bold—the order of the nesting (at least in this case) isn’t particularly important.
4. Unless you’re a supernaturally smooth typist, markup can be a real pain to add to content. Not only is it physically cumbersome to type the < and > symbols, it’s also mentally challenging to keep focused on producing the content and not the markup in which it’s contained.

The devil in the details

I mentioned above that this convention of “all markup is contained in the special characters `<` and `>`” isn’t entirely foolproof. It’s true that these special characters are seldom used in normal text; it’s equally true that they’re on the keyboard for a reason—they are *sometimes* used in normal text.

So when a browser encounters a “real” `<` symbol without a matching `>`, how does it know not to keep scanning the document all the way to the end, looking for the missing `>` symbol? There are ways around this dilemma too, and we’ll see some of them later in this book. Ultimately, though, it’s a real hall of mirrors; you just have to throw up your hands at some point and say, “I don’t care about further exceptions to the exceptions to the exceptions.”

Do you need to know anything about SGML or HTML to start working with XML immediately? No. Much of the background (historical and technological) will be *useful* for you to know, but none of it is required.

The XML Difference

A markup cartoon

Editorial cartoons use a common convention to help make their metaphors easily grasped by their audience: Any object in the drawing that's not obviously a caricature of a familiar personage, a famous building, or a standard editorial-cartoon symbol (such as donkeys for Democrats, Uncle Sam for the USA, and so on) is *labeled*. I want you now to picture yourself in such a cartoon, an animated one.

You're in the lobby of the Ritz-Markup Hotel, standing before an elevator whose doors part to reveal three passengers. Their gender doesn't matter at all, but let's assume for the sake of illustration that they're men. (Cartoonists do everything for the sake of illustration.) You get on the elevator. And as you usually do, you peek discreetly at the three guys sharing your space.

Passenger #1 is apparently an employee of this tony establishment, the elevator operator. The man has the posture of a drill sergeant and radiates an aura of someone who goes about his business with supreme confidence. You observe that he is dressed immaculately—not in a uniform, as you might expect an elevator operator to be—but in a dark, pin-striped three-piece suit. His vest is made of fine silk; a watch chain runs across the front of it. At least you *thought* it was a watch chain, but then you notice that every time the elevator stops at a floor, the man pulls from the watch pocket of his vest a Swiss army knife and unfolds a different blade, which he then inserts into a slot in the elevator control panel and turns, causing the doors to open, pause, and close. Rather oddly, he has an umbrella hooked over one wrist. (Well, who knows? It *might* rain in an elevator.) There's a sign hanging on this man's chest; it reads "SGML."

Passenger #2 could, in a dim room, be mistaken for the elevator operator. Maybe they're related; they share the same cheekbones, the same eyes, and are about the same height. But there's something just a little off-kilter about this fellow. He's not wearing a suit, for starters, but a sport coat and slacks. He slouches against the back wall of the elevator. He's wearing sunglasses, and his left and right socks are of slightly different shades (although they could be mistaken for identical). He bends down to tie a shoe, giving you a good glance at the sign on his back—"HTML," it reads—and also at the shirt tail inelegantly protruding from beneath the hem of his jacket.

Like the second passenger, the third shares certain of Passenger #1's features, the same eyes and cheekbones. This one—the youngest of the three—

stands as erect as the first, though, and also wears a dark suit. But it's a two-piece, *sans* vest, and consequently he has no Swiss army knife on a chain. No umbrella, either. And unlike passenger #2, there are no shirt tails hanging out, and no untied shoes. At one point on the long ride, passenger #3 removes from his jacket pocket a pair of nail scissors, trims off a jagged edge, then puts the scissors back in his pocket and removes an emery board with which he smoothes the trimmed edge. One separate tool for each separate task. Looking more closely—you really are a snoop—you see that on each of his sleeves is a label: "XML."

Meanwhile, back in the real world...

The differences among SGML, HTML, and XML are like those among the three guys in the elevator.

SGML calls the shots for all three, deciding where they will stop and how they will stop there. SGML is ready for every eventuality it might face, even the most unlikely (the thunderstorm in the elevator car). It is formal, rigorous, precise, and darned near complete.

Second-generation markup—HTML—shares a number of its daddy's features. It's very well-suited for casual purposes. You can bring HTML to a business meeting, but it will always look a bit *déclassé*. The freedom to have loose ends dangling here and there means that HTML can get ready for work a lot faster than either SGML or XML, but it also means that on the way out of the elevator door, things have a tendency to snag.

Finally, XML. Its general form and shape are those of SGML. But unless it's raining, it doesn't bring an umbrella along; and if it needs to attend to some personal grooming, it's got the tool that does exactly that (and no more). Its middle-of-the-road attire means that you can bring it along for just about any business need.

As you'll see in Chapter 2, an XML document doesn't look all that different from its SGML or HTML counterparts. Both the minor differences and more numerous similarities are a direct result of the XML framers' experiences with the two earlier markup languages. From the start, the XML specification has included the following ten design principles; the headings are taken straight from the specification document itself:

XML should be straightforwardly usable over the Internet.

This requirement seems almost to go without saying; of course you'd want a new markup language to be usable on the Internet, particularly the World Wide Web.

But “straightforwardly” adds a subtle extra layer of meaning: like HTML, XML is not rocket science. Furthermore—perhaps less obviously—delivering XML documents does not require any change to the network itself, or to its supporting software and protocols.

XML shall support a wide variety of applications.

Again, this seems like an obvious goal. It marks a radical departure for XML versus its HTML cousin, however. The latter’s design supports on its own just a single (albeit powerful) application: Web browsing. All the other stuff that occurs on the Web—Java and ECMAScript, Common Gateway Interface (CGI) forms and database processing, animated images, and audio and video playback—occurs inside simple or exotic add-ons to the core HTML specification itself.

Note that the XML spec does not say *what* other applications will be supported. The XML FAQ, in a favorite burst of encyclopedic whimsy, mentions “music, chemistry, electronics, hill-walking, finance, surfing, petroleum geology, linguistics, cooking, knitting, stellar cartography, history, engineering, rabbit-keeping, mathematics, etc.” as possible applications of XML. All of those can be “supported” by HTML as well, of course, but the difficulty is that HTML forces them all into clothing cut from the same fabric.

XML shall be compatible with SGML.

The intention here is to capitalize on the success of SGML. While it has not been trumpeted in the popular media nearly as much (or as loudly) as HTML, SGML is a critical technological weapon in the arsenal of many industries, from newspapers to banking. If XML can take advantage of the embedded base of SGML authors and—especially—software, it will speed the acceptance of the new markup language by those important customers.

Aside from the existing SGML tools and expertise that can be readily adapted to XML, there is much inherent power in SGML that was not carried forward into the HTML specification. In HTML, for example, you can’t deviate from the set of element types that are specified for the language. XML’s ability to include any element types that a particular purpose requires—the “Extensible” in the name—is a direct descendant of SGML’s.

It shall be easy to write programs which process XML documents.

Note that here we're not talking about coding XML documents themselves, but rather about the *programs which process XML documents.* This objective says that the rules of the language should be simple, not just for humans but for machines as well.

Simple examples abound in HTML of how markup languages can be difficult for software to process, while easy for humans to interpret. For example, the "forgivingness" of the HTML specification—or rather of the way in which it's interpreted by Web browsers—allows many elements to be sloppily nested within one another. This is a generous gift to fumble-fingered Webmasters, but can complicate a software developer's life enormously, especially when there are elements within elements within elements: How does the program know when to "turn off" italics that are signified by a `<bold>` tag, which in turn overlaps a particular font specification, which sort of but not quite exactly falls within the scope of a bulleted list item? All the exception handling—the need to deal intelligently with all those dangling shirt tails—can give a programmer fits.

At one time, XML's developers reportedly asserted that writing an XML application program should be at most a "two-week" project for graduate students. This didn't make it into the specification (maybe they floated it among some graduate students, who threatened a walkout), but the philosophy behind that specific target remains very much a part of XML as formally defined.

One important reason why XML programs need to be simple to write lies in the very extensibility of the language. If you've got a particular XML flavor that has been tailored specifically for genealogical records, for example, it would be great to have a "genealogical XML browser" that knows—far more than a generic Web browser could—exactly where on the user's screen to place birthdates, photos of family members, and so on. The requirements for such a browser would be very different from those for a browser of library catalog entries. If you want to encourage the development of such special-purpose browsers, you've got to keep their requirements simple.

The number of optional elements in XML is to be kept to the absolute minimum, ideally zero.

While XML is a direct descendant of SGML (much more so than HTML), it does away with hundreds of optional "features" added to its parent over the course of many years. (One common way of thinking of XML is as "SGML Lite"—or, as one reference says, "SGML^{-}" rather than "HTML^{++}.")

Note that the "optional features" referred to here are those in the language specification itself. Individual XML document types—the genealogical one I mentioned above, FlixML, the Chemical Markup Language, and so on—can be as baroque and fully featured as their designers desire, including many optional element types *within each document type*.

XML documents should be human-legible and reasonably clear

This, again, is partially a matter of processing efficiency as well as human comfort.

It's important, yes, that you and I be able to read a Triffids.xml file in its "naked," raw-text form, without any fancy software, and figure out what a reviewer of *Day of the Triffids* has to say about the movie, who its cast was, what studio produced it, how likely it is that the Earth would be overrun by hordes of carnivorous ferns, and so on. (If we're writing the review ourselves, it may be equally important to see all the nuts-and-bolts of the markup should the review not "behave" exactly as we'd expected in our FlixML browser.)

But as I mentioned earlier in this chapter, putting not only the content but also the markup itself in plain text form makes the resulting document not only smaller, but more portable across computing platforms. This is a wonderful byproduct of using plain text.

The XML design should be prepared quickly

Internet time, they say, is faster than real time. I don't think I've ever heard an estimate of *how much* faster. Even the seven-times-faster of dog years doesn't seem fast enough, though. A new technology crops up for performing Task X, and a week later three or four competitors appear; the next week, there's a new technology for performing the same task, plus Y and Z.

On the Web, what was about to happen was that large enterprises—corporations, banks, the government, and all the rest of the usual suspects—were starting to wonder if they should bail out of the whole standards process. HTML can do quite a bit, especially with the use of browser plug-ins and the like; but the Web was in danger of collapsing under the combined weight of all the things that HTML *can't* do (or do easily), and of all the proprietary software needed to make HTML stand on its head and do the required backflips.

"Quickly" was never defined, but a sense of urgency underlay the preparation of the XML specification. Logistics could have complicated matters further—people working on the spec were widely separated geographically, and most had other commitments to their employers, schools, and so on (to say nothing of their personal lives). Fortunately they all were able to take advantage of advanced technology to communicate with one another, in e-mail, video, and phone conferences. Within about a year of their first "meeting," the next-to-final version of the language spec had been posted on the site of the World Wide Web Consortium. (The consortium, known as the W3C, is an arbiter of specifications and standards for new Internet technologies.) Given the complexities they were dealing with, this was remarkable.

The design of XML shall be formal and concise

Okay, here's this new markup language. Its stated goal is, in brief, to simplify the task of delivering complex content over the Web. How embarrassing it would have been if the design of the new language were more complicated and ambiguous than the language itself!

The XML specification is both formal and concise because it is written in something called Extended Backus-Naur Format, or EBNF. EBNF, a common tool for declaring the syntax of new programming languages, will probably never win any prizes for aesthetic appeal, except among people who appreciate engineering beauty. For example, if you asked a fairly simple question such as, "What's the formal definition of the XML term 'children'?," the spec would answer you as follows:

```
[47] children ::= (choice | seq) ('?' | '*' | '+')?
```

To say that this is alarming doesn't begin to do justice to the term "alarm." But it's undeniably efficient, unambiguous, and yes, even beautiful (in the same way that the interior of Arnold Schwarzenneger's forearm was in *The Terminator*).

Don't panic. *You don't have to know EBNF in order to know XML.* It's true that you can't escape it if you want to read and understand the formal language specification; if you're satisfied with simply using XML, though, you needn't give EBNF a second thought.

(By the way, none of the HTML coders I've ever met knows anything about EBNF. But the HTML spec is written in EBNF, too.)

Ch01.fm Page 16 Tuesday, July 18, 2000 10:39 AM

XML documents shall be easy to create

Like "quickly," "straightforwardly," "legible," and so on, "easy to create" isn't spelled out. But I have to admit that I wouldn't be comfortable if this *wasn't* a goal of XML, no matter how vague the term.

As I mentioned earlier in this chapter, you don't need anything other than a computer to "write XML": no special word processor or other software is required. You can even make do with pencil and paper as long as you either: (a) don't care to actually put your XML online; or (b) have someone else to transcribe your chicken-scratchings into a text file.

Terseness in XML markup is of minimal importance

SGML and HTML both provide for *omitting* markup—particularly end-tags—in some cases. One of the reasons why they permit this is that the resulting documents are shorter and more concise.

The cost of this terseness is often clarity, determining what is intended at a given point where the optional markup might be expected. (It's rather like one of those ancient fortune-tellers trying to predict the future by examining a single entrail.) And the drafters of the XML spec set clarity as one of their guiding principles—hence, this design goal.

Ulterior motives

One of my favorite XML references is the Annotated XML Specification, at:

```
www.xml.com/axml/axml.html
```

Written by Tim Bray, one of the spec's coeditors, this annotated version includes copious notes, explanations, and asides that illuminate the official requirements in often quite wonderful ways.

In discussing the above ten design goals, Bray mentions two other factors that motivated the spec's writers:

Internationalization: "We bent over backward to make XML work properly in all the world's scripts, with a fair degree of success."

The DPH: DPH is an acronym for *D*esperate *P*erl *H*acker. Perl, as you may know, is one of the premier languages for programming the Web (perhaps *the* premier language for that task). As Bray says, the DPH is "the luckless subordinate who is informed that some global change is required in a large, complex document inventory at short notice, and who is able to deliver by applying a scripting facility such as Perl to cleanly-structured data such as XML."

What XML Isn't

First, foremost, and maybe most confusingly, XML is not *a* single markup language.

I know, I know. It's right there in the name, isn't it? "Extensible Markup Language." But despite certain general rules (the requirement that all elements nest properly, the presence of some common conventions such as how to code comments, and so on), there is no single, broadly useful set of markup to learn in order to learn XML. XML is all about *separate markup languages for separate purposes.* This is a huge departure for anyone who has spent the last several years of his or her life memorizing the vagaries of HTML. It's also why most books about XML, including this one, present XML by way of "demonstrators"—customized markup languages (like FlixML) that show the range of things *possible* with XML but don't teach you XML as such, except by example.

Second, XML is not an "all things to all people" solution. HTML will continue to grow, and it will continue to be both more convenient and more flexible for many purposes. You don't need to throw away everything you know (if you do know) about the difference between the effect of HTML's unordered list tag `<ul>` and that of the ordered list tag `<ol>`, for instance.

Third, XML is not in itself a display language. In HTML, tags (and the corresponding elements) generally serve two purposes: they both add structure to documents and imply a certain display style. Paragraphs start on a new line; headings vary in size; bold is, well, bold. In itself, XML is almost exclusively a language not for defining how things look on a screen, but for defining specific content—a language for manipulating the *what* rather than the *how.*

What XML Is

I'll repeat the point from the last paragraph: On its own, XML is a tool for manipulating the *what* rather than the *how.*

It's natural for people who have used the Web at all—heck, even just readers of magazines—to assume the importance of what font sizes and typefaces headings will be displayed in, whether cells in a table will be centered or left-justified, what color the page background will be, and so on.

We've been spoiled by later developments in HTML, though. Originally, for instance, there were no tags for specifying bold or italics text—there were just `<strong>` and `<em>` (for *emphasized*) tags. It was up to a browser vendor to decide what to do with those tags in an HTML document; if they wanted to, they could render all `<strong>` text in all uppercase letters, and all `<em>` as underlined. This made page designers crazy: By God, if they *meant* italics, then the browser better not display non-italicized capital letters!

So over time, HTML has drifted into a stew of combined structural and display markup. Most recently, with the introduction of the Cascading Style Sheets standard (CSS1 originally, and since May, 1998, CSS2, with CSS3 in the works as of this writing), a whole mechanism for controlling the *how* has been sort of nailed to the side of the bubbling HTML pot. In the meantime, especially with all the multimedia extensions to Web pages (and the resulting media hoopla), the emphasis on document structure has been lost.

But even if you're an HTML purist who'd die before sullying your pages with font changes, the *structures* possible with that language aren't really sufficient for many advanced (even simple) purposes. They're a least-common-denominator set of structures that could be used in documents of any kind. There's no difference in the *meaning* of the term "paragraph" as it's used on *CNN Interactive* and its meaning on a Web site of academic treatises on the genetic manipulation of the common housefly.[2]

I'll give you another analogy—this one from a different side of computing.

In everyday life I'm an application developer, specifically of small-scale networked databases being used by up to about ten people at a time. One difficulty in designing a new database always is the need to help potential users understand why I don't want to use a spreadsheet to do their work for them, but insist on a real database instead.

First, consider the spreadsheet. Aside from its roots in accounting—the classic row-and-column format of a ledger—every spreadsheet just makes *sense* to people. They're used to seeing printed reports, for example, with one row per "thing" and one column for each of the thing's characteristics.

But then I point out the variety of uses to which they propose to put their data. They don't just want a single report format. They sometimes want to see everything about everything (in which case, a spreadsheet-like layout can indeed

2. There was a profile of Michael Jordan as a celebrity advertising icon a while back in *The New Yorker* magazine. On the face of it, the article has nothing at all to do with XML vs. HTML. But there's a wonderful line by the author (Henry Louis Gates, Jr.) that expresses this quite succinctly: "Different celebrities were repositories of different values and associations: Sigourney Weaver didn't *mean* the same thing as Loni Anderson."

make sense, even if the resulting printout is fifty yards wide and only eight inches deep); more often, they want just to see everything about *one* thing (in which case using a spreadsheet is just crazy). Sometimes they want to highlight a feature that all the things have in common. They don't want to have to type in a lengthy text string more than once. And so on.

It doesn't usually take more than a half-hour of this to convince them.

HTML is like a spreadsheet. Every single HTML document is, basically, the same structure. You can dress it up with colors, fonts, style sheets, or however you want, but it's still the equivalent of a mindless row-and-column design.

XML, by contrast, is like a relational database. The structure is optimized for the particular application. The structure isn't mindless; it's actually mind*ful*.

A truism of Web-page authorship is, "Content is king." The theory is that if you dump rich enough content into a page, the page's value goes up proportionately. But not all truisms are *true*, and the content-is-king theory has in my opinion turned out to be one of the big lies of the technological age—*because HTML lets you emphasize style over substance.*

Ain't no such thing in XML. XML is 100% about content, and how it's structured internally to be most useful.

From the Sublime to the Ridiculous

Well, there's content, and there's content.

Much of the attention paid to XML thus far has been in terms of its potential as a heavyweight application tool. If you're a doctor (we're told repeatedly), you can use XML for maintaining patient records, and then you can make those records accessible and understandable to the patients themselves simply by viewing the data in a "patient's-eye view of his or her records" browser. Corporations can develop intranet databases with `<invoice>`, `<partnumber>`, `<custnumber>`, `<custname>`, and—who knows?—`<warrantyexpirationdate>` tags, and they can easily share those databases with other corporations (such as insurance and service companies).

Yeah, you can do all that stuff with XML. But I don't think you need to be totally serious about it. Commerce, after all, is long, but life is short. You can use XML for almost anything…even a piece of fluff like a B movie database.

Just FlixML

Historically, the Bs—also called "programmers," the implications of which we probably don't want to think about too much—were commonly shown as filler in a bill with more big-ticket films. In an indoor theater, the B might have appeared first, as a warmup for the real feature. In a drive-in theater, it might have been shown second or third, when the audience who had stayed for it was, let's say, less interested in the content of what was on the screen than in the fact that it was dark, and that they were sharing an enclosed, semi-private space with a member of the opposite sex.

By way of analogy, think of the warmup act in a concert.[3] Many in the audience will have no particular reason for hearing the warmup act, may arrive late precisely in order to miss it, or may ignore it if present. But sometimes, even an otherwise dreadful warmup band will come up with an absolute miracle of music-making, one that leaves the headline group's own performance seem pale by comparison.

That happens with B movies, too. Most of them you'll never be poorer for having missed, but many are gems that went unnoticed at the time *because* of their having been paired with blockbusters. They got lost in the shuffle. (And if the A picture was itself a dog, the B was doubly damned.)

Unlike concert warmup acts, a lot of the best B movies are still accessible to us in some way—thanks to television and videotape. But how do we know which ones to go after? How do we tell other enlightened souls about the best ones, in a way that's consistent, complete, reliable, and—ideally—fun?

We use, of course, just FlixML.

The nature of the beast

I'll have much more to say about why FlixML includes the specific things it does (and leaves others out) in Part 4, "Rolling Your Own XML Application." But here are the general areas I want it to cover:

1. The facts: FlixML must be capable of holding as much objective information about a B movie as possible. This stuff is pretty dry—title, year first released, cast, crew, studio, and so on.

3. If you're old enough, you can also think of the "flip side" of old 45-rpm records. This was where the artist or producer dumped a second-tier song to complement the (presumed) hit on the other side. The flip side was also called the B side.

2. The story: This is a little more freeform. A potential viewer of a B film needs to know what it's about.
3. The quality: Not all low-budget movies are created equal, even if they're identical in most of their objective dimensions.

So I've set up FlixML not just to describe a given movie in factual terms, but also to *characterize* it. There are provisions for including reviews (both your own, and those of others—including real critics).

But a markup language for describing movies in general—not just Bs—would cover those same dimensions. What would be the additional features that a B-movie-specific markup language should include?

To answer that question, I've provided for ratings that are both finer-tuned and more subjective, culminating in a single rating on a "B-ness scale." This B-ness figure captures, in a nutshell, how much a film should be considered a true B film, versus a true turkey, an A film that's simply scarred by subpar production values, and so on.

Which brings us to the question: What exactly are the attributes of B movies versus all others?

I've talked to friends about this notion. I know that B movies, like As and Cs, have a director, a cinematographer, and a cast; that they're in black-and-white or color; that they're silent or sound—I know all that. But what are the essential ingredients of a B movie?

Here are some of the things we've come up with:

- Little or no redeeming social, artistic, or intellectual value
- Lukewarm commercial success
- Cheesy or at least obvious "special" effects
- Recycled plot lines
- Don't have sequels (although sequels to some other classifications of movies are themselves often Bs)
- No "name" stars or directors (at least at the time the movie was made)
- TV (and later, videotape) saved them from certain oblivion
- You'll never find a B movie's soundtrack on CD
- There are no parodies of specific B movies
- Not shown on primetime network television
- The men wear suits 24 hours a day, the women dresses, the kids either short pants with suspenders or pinafores
- Nobody ever eats a complete meal

- If you fall asleep during a B movie that is followed by another B movie, and then wake up, it takes you a half-hour to realize you're watching a different movie

(No doubt you can come up with some of your own criteria, but this is a good starting point.)

The key feature that distinguishes a B movie from any other can probably be stated simply like this: It has no *extrinsic* qualities that would make you want to see it. Someone might pass along a word-of-mouth recommendation, but you otherwise probably happened on it by accident. You've read no reviews of it; you don't know any other movies by the same director; and while certain cast members later may have become "stars," you wouldn't have known them at the time. All of a B movie's virtues, if any, are *intrinsic*.

So having a computer handy as you proceed through *Just XML* will be important, of course…just remember to keep the VCR warmed up and ready as well. And your tongue planted firmly in cheek (I know that's where mine will be).

B Alert!

Watch This Space

Throughout *Just XML*, keep your eyes open for a box such as this one.

Each *B Alert!* box will contain a capsule description of a classic B film. I'll include notes on the year the film was released, its plot, and especially why I think it's worth hunting down a copy of the film for your own viewing.

As in most other such, er, artistic pursuits, there will be of course a certain amount of subjectivity at work here. A flick I recommend may well leave you scratching your head, or—who knows?—out-and-out repulsed. (That's always a possibility when looking through films that got lackluster studio support during their making. On the other hand, it's always a possibility when looking through films that *did* receive a lot of TLC and promotional attention from their studios. Go figure.) I'll try at least, though, to make the pitch both clear and encouraging enough for you to make an informed judgment of your own.

A few bits of XML code in *Just XML* describe movies that don't exist. Most of them, though, are real enough. And most of *those* will be honored (if that's not stretching a term too far) with their own *B Alert!* boxes.

One last thing: At my Web site, I've got an XML tutorial, links to XML-related sites, information about B movies, and several examples of FlixML documents both in raw XML format and converted into HTML for viewing. Please stop in; the address is:

```
www.flixml.org
```