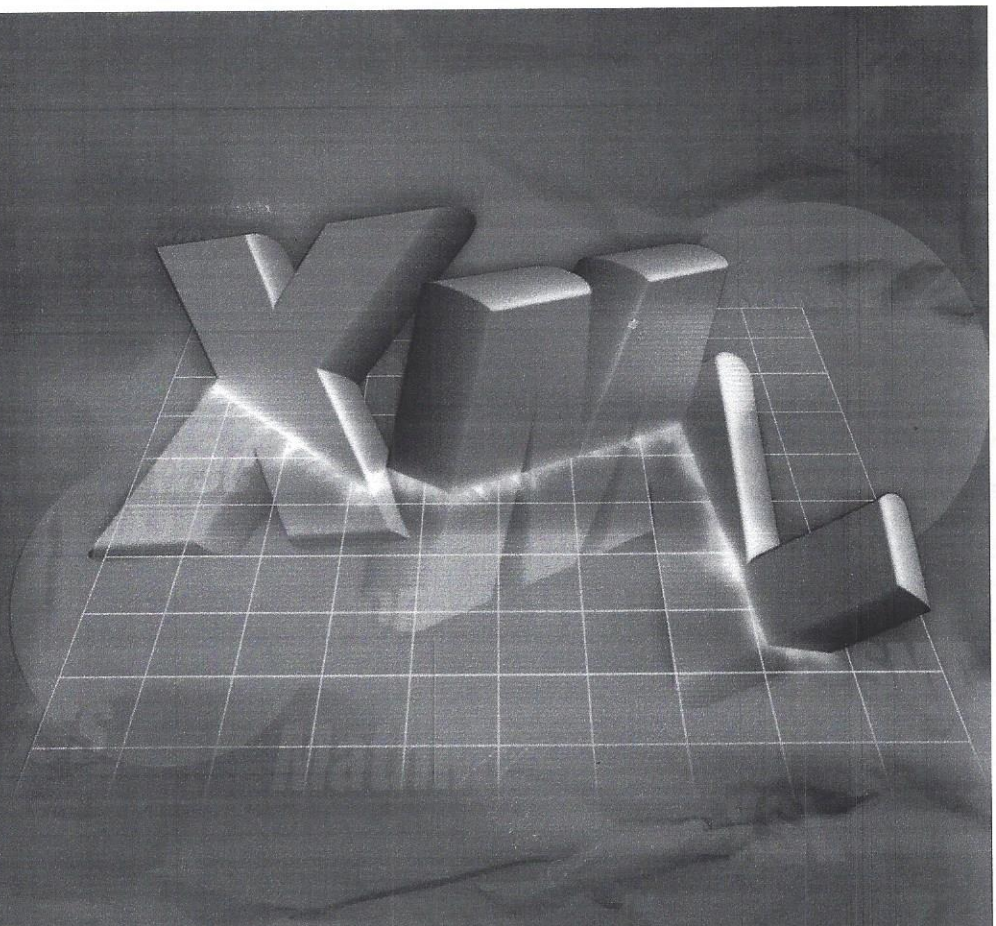
100
ONE HUNDRED PERCENT

COMPREHENSIVE
AUTHORITATIVE
WHAT YOU NEED
ONE HUNDRED PERCENT

Master the simple yet powerful new markup language that's revolutionizing the Web

well-formed, sensibly organized Web documents

Create entirely new markup languages to fit your own needs



XML™ Bible

CD-ROM INSIDE!

- Code for every numbered listing in the book and additional examples
- XML browsers and tools
- Relevant W3C standards

Elliotte Rusty Harold

XML™ Bible

Elliotte Rusty Harold

IDG Books Worldwide, Inc.
An International Data Group Company

Foster City, CA ✦ Chicago, IL ✦ Indianapolis, IN ✦ New York, NY

XML™ Bible

Published by
IDG Books Worldwide, Inc.
An International Data Group Company
919 E. Hillsdale Blvd., Suite 400
Foster City, CA 94404
www.idgbooks.com (IDG Books Worldwide Web site)

ISBN: 0-7645-3236-7

Printed in the United States of America

10 9 8 7 6 5 4 3

1O/QV/QY/ZZ/FC

Distributed in the United States by IDG Books Worldwide, Inc.

Distributed by CDG Books Canada Inc. for Canada; by Transworld Publishers Limited in the United Kingdom; by IDG Norge Books for Norway; by IDG Sweden Books for Sweden; by IDG Books Australia Publishing Corporation Pty. Ltd. for Australia and New Zealand; by TransQuest Publishers Pte Ltd. for Singapore, Malaysia, Thailand, Indonesia, and Hong Kong; by Gotop Information Inc. for Taiwan; by ICG Muse, Inc. for Japan; by Norma Comunicaciones S.A. for Colombia; by Intersoft for South Africa; by Eyrolles for France; by International Thomson Publishing for Germany, Austria and Switzerland; by Distribuidora Cuspide for Argentina; by Livraria Cultura for Brazil; by Ediciones ZETA S.C.R. Ltda. for Peru; by WS Computer Publishing Corporation, Inc., for the Philippines; by Contemporanea de Ediciones for Venezuela; by Express Computer Distributors for the Caribbean and West Indies; by Micronesia Media Distributor, Inc. for Micronesia; by Grupo Editorial Norma S.A. for Guatemala; by Chips Computadoras S.A. de C.V. for Mexico; by Editorial Norma de Panama S.A. for Panama; by American Bookshops for Finland. Authorized Sales Agent: Anthony Rudkin Associates for the Middle East and North Africa.

For general information on IDG Books Worldwide's books in the U.S., please call our Consumer Customer Service department at 800-762-2974. For reseller information, including discounts and premium sales, please call our Reseller Customer Service department at 800-434-3422.

For information on where to purchase IDG Books Worldwide's books outside the U.S., please contact our International Sales department at 317-596-5530 or fax 317-596-5692.

For consumer information on foreign language translations, please contact our Customer Service department at 800-434-3422, fax 317-596-5692, or e-mail rights@idgbooks.com.

For information on licensing foreign or domestic rights, please phone +1-650-655-3109.

For sales inquiries and special prices for bulk quantities, please contact our Sales department at 650-655-3200 or write to the address above.

For information on using IDG Books Worldwide's books in the classroom or for ordering examination copies, please contact our Educational Sales department at 800-434-2086 or fax 317-596-5499.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 650-655-3000 or fax 650-655-3299.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 978-750-4470.

Library of Congress Cataloging-in-Publication Data
Harold, Elliote Rusty.
XML bible / Elliote Rusty Harold.
p. cm.
ISBN 0-7645-3236-7 (alk. paper)
1. XML (Document markup language) I. Title.
QA76.76.H94H34 1999 99-31021
005.7'2–dc21 CIP

ABOUT IDG BOOKS WORLDWIDE

Welcome to the world of IDG Books Worldwide.

IDG Books Worldwide, Inc., is a subsidiary of International Data Group, the world's largest publisher of computer-related information and the leading global provider of information services on information technology. IDG was founded more than 30 years ago by Patrick J. McGovern and now employs more than 9,000 people worldwide. IDG publishes more than 290 computer publications in over 75 countries. More than 90 million people read one or more IDG publications each month.

Launched in 1990, IDG Books Worldwide is today the #1 publisher of best-selling computer books in the United States. We are proud to have received eight awards from the Computer Press Association in recognition of editorial excellence and three from Computer Currents' First Annual Readers' Choice Awards. Our best-selling ...*For Dummies*® series has more than 50 million copies in print with translations in 31 languages. IDG Books Worldwide, through a joint venture with IDG's Hi-Tech Beijing, became the first U.S. publisher to publish a computer book in the People's Republic of China. In record time, IDG Books Worldwide has become the first choice for millions of readers around the world who want to learn how to better manage their businesses.

Our mission is simple: Every one of our books is designed to bring extra value and skill-building instructions to the reader. Our books are written by experts who understand and care about our readers. The knowledge base of our editorial staff comes from years of experience in publishing, education, and journalism — experience we use to produce books to carry us into the new millennium. In short, we care about books, so we attract the best people. We devote special attention to details such as audience, interior design, use of icons, and illustrations. And because we use an efficient process of authoring, editing, and desktop publishing our books electronically, we can spend more time ensuring superior content and less time on the technicalities of making books.

You can count on our commitment to deliver high-quality books at competitive prices on topics you want to read about. At IDG Books Worldwide, we continue in the IDG tradition of delivering quality for more than 30 years. You'll find no better book on a subject than one from IDG Books Worldwide.

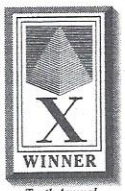
IDG BOOKS WORLDWIDE

John Kilcullen
Chairman and CEO
IDG Books Worldwide, Inc.

Steven Berkowitz
President and Publisher
IDG Books Worldwide, Inc.

Eighth Annual Computer Press Awards 1992

Ninth Annual Computer Press Awards 1993



Tenth Annual Computer Press Awards 1994

Eleventh Annual Computer Press Awards 1995

IDG is the world's leading IT media, research and exposition company. Founded in 1964, IDG had 1997 revenues of $2.05 billion and has more than 9,000 employees worldwide. IDG offers the widest range of media options that reach IT buyers in 75 countries representing 95% of worldwide IT spending. IDG's diverse product and services portfolio spans six key areas including print publishing, online publishing, expositions and conferences, market research, education and training, and global marketing services. More than 90 million people read one or more of IDG's 290 magazines and newspapers, including IDG's leading global brands — Computerworld, PC World, Network World, Macworld and the Channel World family of publications. IDG Books Worldwide is one of the fastest-growing computer book publishers in the world, with more than 700 titles in 36 languages. The "...For Dummies®" series alone has more than 50 million copies in print. IDG offers online users the largest network of technology-specific Web sites around the world through IDG.net (http://www.idg.net), which comprises more than 225 targeted Web sites in 55 countries worldwide. International Data Corporation (IDC) is the world's largest provider of information technology data, analysis and consulting, with research centers in over 41 countries and more than 400 research analysts worldwide. IDG World Expo is a leading producer of more than 168 globally branded conferences and expositions in 35 countries including E3 (Electronic Entertainment Expo), Macworld Expo, ComNet, Windows World Expo, ICE (Internet Commerce Expo), Agenda, DEMO, and Spotlight. IDG's training subsidiary, ExecuTrain, is the world's largest computer training company, with more than 230 locations worldwide and 785 training courses. IDG Marketing Services helps industry-leading IT companies build international brand recognition by developing global integrated marketing programs via IDG's print, online and exposition products worldwide. Further information about the company can be found at www.idg.com.

1/24/99

Contents

Part V: XML Applications 655

Chapter 20: Reading Document Type Definitions 657

PART

I

Introducing XML

In This Part

CHAPTER 1

An Eagle's Eye View of XML

In This Chapter

What is XML?

Why are developers excited about XML?

The life of an XML document

Related technologies

This first chapter introduces you to XML. It explains in general what XML is and how it is used. It shows you how the different pieces of the XML equation fit together, and how an XML document is created and delivered to readers.

What Is XML?

XML stands for Extensible Markup Language (often written as eXtensibleMarkup Language to justify the acronym). XML is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. It is a meta-markup language that defines a syntax used to define other domain-specific, semantic, structured markup languages.

XML Is a Meta-Markup Language

The first thing you need to understand about XML is that it isn't just another markup language like the Hypertext Markup Language (HTML) or troff. These languages define a fixed set of tags that describe a fixed number of elements. If the markup language you use doesn't contain the tag you need—you're out of luck. You can wait for the next version of the markup language hoping that it includes the tag you need; but then you're really at the mercy of what the vendor chooses to include.

XML, however, is a meta-markup language. It's a language in which you make up the tags you need as you go along. These tags must be organized according to certain general principles, but they're quite flexible in their meaning. For instance, if you're working on genealogy and need to describe people, births, deaths, burial sites, families, marriages, divorces, and so on, you can create tags for each of these. You don't have to force your data to fit into paragraphs, list items, strong emphasis, or other very general categories.

The tags you create can be documented in a Document Type Definition (DTD). You'll learn more about DTDs in Part II of this book. For now, think of a DTD as a vocabulary and a syntax for certain kinds of documents. For example, the MOL.DTD in Peter Murray-Rust's Chemical Markup Language (CML) describes a vocabulary and a syntax for the molecular sciences: chemistry, crystallography, solid state physics, and the like. It includes tags for atoms, molecules, bonds, spectra, and so on. This DTD can be shared by many different people in the molecular sciences field. Other DTDs are available for other fields, and you can also create your own.

XML defines a meta syntax that domain-specific markup languages like MusicML, MathML, and CML must follow. If an application understands this meta syntax, it automatically understands all the languages built from this meta language. A browser does not need to know in advance each and every tag that might be used by thousands of different markup languages. Instead it discovers the tags used by any given document as it reads the document or its DTD. The detailed instructions about how to display the content of these tags are provided in a separate style sheet that is attached to the document.

For example, consider Schrodinger's equation:

$$i\hbar\frac{\partial\psi(r,t)}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi(r,t)}{\partial x^2} + V(r)\,\psi(r,t)$$

Scientific papers are full of equations like this, but scientists have been waiting eight years for the browser vendors to support the tags needed to write even the most basic math. Musicians are in a similar bind, since Netscape Navigator and Internet Explorer don't support sheet music.

XML means you don't have to wait for browser vendors to catch up with what you want to do. You can invent the tags you need, when you need them, and tell the browsers how to display these tags.

XML Describes Structure and Semantics, Not Formatting

The second thing to understand about XML is that XML markup describes a document's structure and meaning. It does not describe the formatting of the elements on the page. Formatting can be added to a document with a style sheet. The document itself only contains tags that say what is in the document, not what the document looks like.

- By contrast, HTML encompasses formatting, structural, and semantic markup. <B> is a formatting tag that makes its content bold. <STRONG> is a semantic tag that means its contents are especially important. <TD> is a structural tag that indicates that the contents are a cell in a table. In fact, some tags can have all three kinds of meaning. An <H1> tag can simultaneously mean 20 point Helvetica bold, a level-1 heading, and the title of the page.

  For example, in HTML a song might be described using a definition title, definition data, an unordered list, and list items. But none of these elements actually have anything to do with music. The HTML might look something like this:

```
<dt>Hot Cop
<dd> by Jacques Morali, Henri Belolo, and Victor Willis
<ul>
<li>Producer: Jacques Morali
<li>Publisher: PolyGram Records
<li>Length: 6:20
<li>Written: 1978
<li>Artist: Village People
</ul>
```

  In XML the same data might be marked up like this:

```
<SONG>
  <TITLE>Hot Cop</TITLE>
  <COMPOSER>Jacques Morali</COMPOSER>
  <COMPOSER>Henri Belolo</COMPOSER>
  <COMPOSER>Victor Willis</COMPOSER>
  <PRODUCER>Jacques Morali</PRODUCER>
  <PUBLISHER>PolyGram Records</PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
```

  Instead of generic tags like <dt> and <li>, this listing uses meaningful tags like <SONG>, <TITLE>, <COMPOSER>, and <YEAR>. This has a number of advantages, including that it's easier for a human to read the source code to determine what the author intended.

  XML markup also makes it easier for non-human automated robots to locate all of the songs in the document. In HTML robots can't tell more than that an element is a dt. They cannot determine whether that dt represents a song title, a definition, or just some designer's favorite means of indenting text. In fact, a single document may well contain dt elements with all three meanings.

  XML element names can be chosen such that they have extra meaning in additional contexts. For instance, they might be the field names of a database. XML is far more flexible and amenable to varied uses than HTML because a limited number of tags don't have to serve many different purposes.

Why Are Developers Excited about XML?

XML makes easy many Web-development tasks that are extremely painful using only HTML, and it makes tasks that are impossible with HTML, possible. Because XML is eXtensible, developers like it for many reasons. Which ones most interest you depend on your individual needs. But once you learn XML, you're likely to discover that it's the solution to more than one problem you're already struggling with. This section investigates some of the generic uses of XML that excite developers. In Chapter 2, you'll see some of the specific applications that have already been developed with XML.

Design of Domain-Specific Markup Languages

XML allows various professions (e.g., music, chemistry, math) to develop their own domain-specific markup languages. This allows individuals in the field to trade notes, data, and information without worrying about whether or not the person on the receiving end has the particular proprietary payware that was used to create the data. They can even send documents to people outside the profession with a reasonable confidence that the people who receive them will at least be able to view the documents.

Furthermore, the creation of markup languages for individual domains does not lead to bloatware or unnecessary complexity for those outside the profession. You may not be interested in electrical engineering diagrams, but electrical engineers are. You may not need to include sheet music in your Web pages, but composers do. XML lets the electrical engineers describe their circuits and the composers notate their scores, mostly without stepping on each other's toes. Neither field will need special support from the browser manufacturers or complicated plug-ins, as is true today.

Self-Describing Data

Much computer data from the last 40 years is lost, not because of natural disaster or decaying backup media (though those are problems too, ones XML doesn't solve), but simply because no one bothered to document how one actually reads the data media and formats. A Lotus 1-2-3 file on a 10-year old 5.25-inch floppy disk may be irretrievable in most corporations today without a huge investment of time and resources. Data in a less-known binary format like Lotus Jazz may be gone forever.

XML is, at a basic level, an incredibly simple data format. It can be written in 100 percent pure ASCII text as well as in a few other well-defined formats. ASCII text is reasonably resistant to corruption. The removal of bytes or even large sequences of bytes does not noticeably corrupt the remaining text. This starkly contrasts with many other formats, such as compressed data or serialized Java objects where the corruption or loss of even a single byte can render the entire remainder of the file unreadable.

At a higher level, XML is self-describing. Suppose you're an information archaeologist in the 23rd century and you encounter this chunk of XML code on an old floppy disk that has survived the ravages of time:

```
<PERSON ID="p1100" SEX="M">
  <NAME>
    <GIVEN>Judson</GIVEN>
    <SURNAME> McDaniel</SURNAME>
  </NAME>
  <BIRTH>
    <DATE>21 Feb 1834</DATE>  </BIRTH>
  <DEATH>
    <DATE>9 Dec 1905</DATE>  </DEATH>
</PERSON>
```

Even if you're not familiar with XML, assuming you speak a reasonable facsimile of 20th century English, you've got a pretty good idea that this fragment describes a man named Judson McDaniel, who was born on February 21, 1834 and died on December 9, 1905. In fact, even with gaps in, or corruption of the data, you could probably still extract most of this information. The same could not be said for some proprietary spreadsheet or word-processor format.

Furthermore, XML is very well documented. The W3C's XML 1.0 specification and numerous paper books like this one tell you exactly how to read XML data. There are no secrets waiting to trip up the unwary.

Interchange of Data Among Applications

- Since XML is non-proprietary and easy to read and write, it's an excellent format for the interchange of data among different applications. One such format under current development is the Open Financial Exchange Format
- (OFX). OFX is designed to let personal finance programs like Microsoft Money and Quicken trade data. The data can be sent back and forth between programs and exchanged with banks, brokerage houses, and the like.

OFX is discussed in Chapter 2.

As noted above, XML is a non-proprietary format, not encumbered by copyright, patent, trade secret, or any other sort of intellectual property restriction. It has been designed to be extremely powerful, while at the same time being easy for both human beings and computer programs to read and write. Thus it's an obvious choice for exchange languages.

By using XML instead of a proprietary data format, you can use any tool that understands XML to work with your data. You can even use different tools for different purposes, one program to view and another to edit for instance. XML keeps you from getting locked into a particular program simply because that's what

your data is already written in, or because that program's proprietary format is all your correspondent can accept.

- For example, many publishers require submissions in Microsoft Word. This means that most authors have to use Word, even if they would rather use WordPerfect or Nisus Writer. So it's extremely difficult for any other company to publish a competing word processor unless they can read and write Word files. Since doing so requires a developer to reverse-engineer the undocumented Word file format, it's a significant investment of limited time and resources. Most other word processors have a limited ability to read and write Word files, but they generally lose track of graphics, macros, styles, revision marks, and other important features. The problem is that Word's document format is undocumented, proprietary, and constantly changing. Word tends to end up winning by default, even when writers would prefer to use other, simpler programs. If a common word-processing format were developed in XML, writers could use the program of their choice.

Structured and Integrated Data

- XML is ideal for large and complex documents because the data is structured. It not only lets you specify a vocabulary that defines the elements in the document; it also lets you specify the relations between elements. For example, if you're putting together a Web page of sales contacts, you can require that every contact have a phone number and an email address. If you're inputting data for a database, you can make sure that no fields are missing. You can require that every book have an author. You can even provide default values to be used when no data is entered.

- XML also provides a client-side include mechanism that integrates data from multiple sources and displays it as a single document. The data can even be rearranged on the fly. Parts of it can be shown or hidden depending on user actions. This is extremely useful when you're working with large information repositories like relational databases.

The Life of an XML Document

XML is, at the root, a document format. It is a series of rules about what XML documents look like. There are two levels of conformity to the XML standard. The first is *well-formedness* and the second is validity. Part I of this book shows you how to write well-formed documents. Part II shows you how to write valid documents.

HTML is a document format designed for use on the Internet and inside Web browsers. XML can certainly be used for that, as this book demonstrates. However, XML is far more broadly applicable. As previously discussed, it can be used as a storage format for word processors, as a data interchange format for different programs, as a means of enforcing conformity with Intranet templates, and as a way to preserve data in a human-readable fashion.

However, like all data formats, XML needs programs and content before it's useful. So it isn't enough to only understand XML itself which is little more than a specification for what data should look like. You also need to know how XML documents are edited, how processors read XML documents and pass the information they read on to applications, and what these applications do with that data.

Editors

XML documents are most commonly created with an editor. This may be a basic text editor like Notepad or vi that doesn't really understand XML at all. On the other hand, it may be a completely WYSIWYG editor like Adobe FrameMaker that insulates you almost completely from the details of the underlying XML format. Or it may be a structured editor like JUMBO that displays XML documents as trees. For the most part, the fancy editors aren't very useful yet, so this book concentrates on writing raw XML by hand in a text editor.

Other programs can also create XML documents. For example, later in this book, in the chapter on designing a new DTD, you'll see some XML data that came straight out of a FileMaker database. In this case, the data was first entered into the FileMaker database. Then a FileMaker calculation field converted that data to XML. In general, XML works extremely well with databases.

Specifically, you'll see this in Chapter 23, *Designing a New XML Application*.

In any case, the editor or other program creates an XML document. More often than not this document is an actual file on some computer's hard disk, but it doesn't absolutely have to be. For example, the document may be a record or a field in a database, or it may be a stream of bytes received from a network.

Parsers and Processors

An XML parser (also known as an XML processor) reads the document and verifies that the XML it contains is well formed. It may also check that the document is valid, though this test is not required. The exact details of these tests will be covered in Part II. But assuming the document passes the tests, the processor converts the document into a tree of elements.

Browsers and Other Tools

Finally the parser passes the tree or individual nodes of the tree to the end application. This application may be a browser like Mozilla or some other program that understands what to do with the data. If it's a browser, the data will be displayed to the user. But other programs may also receive the data. For instance, the data might be interpreted as input to a database, a series of musical notes to play, or a Java program that should be launched. XML is extremely flex-ible and can be used for many different purposes.

The Process Summarized

To summarize, an XML document is created in an editor. The XML parser reads the document and converts it into a tree of elements. The parser passes the tree to the browser that displays it. Figure 1-1 shows this process.

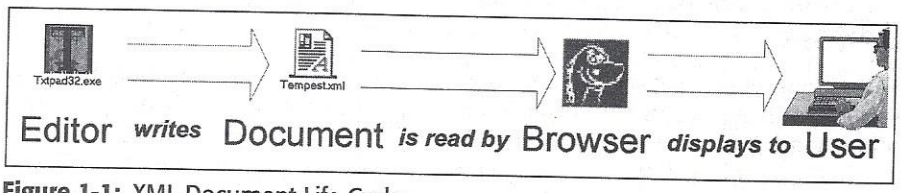


Figure 1-1: XML Document Life Cycle

It's important to note that all of these pieces are independent and decoupled from each other. The only thing that connects them all is the XML document. You can change the editor program independently of the end application. In fact you may not always know what the end application is. It may be an end user reading your work, or it may be a database sucking in data, or it may even be something that hasn't been invented yet. It may even be all of these. The document is independent of the programs that read it.

HTML is also somewhat independent of the programs that read and write it, but it's really only suitable for browsing. Other uses, like database input, are outside its scope. For example, HTML does not provide a way to force an author to include certain required content, like requiring that every book have an ISBN number. In XML you *can* require this. You can even enforce the order in which particular elements appear (for example, that level-2 headers must always follow level-1 headers).

Related Technologies

XML doesn't operate in a vacuum. Using XML as more than a data format requires interaction with a number of related technologies. These technologies include HTML for backward compatibility with legacy browsers, the CSS and XSL style-sheet languages, URLs and URIs, the XLL linking language, and the Unicode character set.

Hypertext Markup Language

Mozilla 5.0 and Internet Explorer 5.0 are the first Web browsers to provide some (albeit incomplete) support for XML, but it takes about two years before most users have upgraded to a particular release of the software. (In 1999, my wife Beth is still

using Netscape 1.1.) So you're going to need to convert your XML content into classic HTML for some time to come.

Therefore, before you jump into XML, you should be completely comfortable with HTML. You don't need to be an absolutely snazzy graphical designer, but you should know how to link from one page to the next, how to include an image in a document, how to make text bold, and so forth. Since HTML is the most common output format of XML, the more familiar you are with HTML, the easier it will be to create the effects you want.

On the other hand, if you're accustomed to using tables or single-pixel GIFs to arrange objects on a page, or if you start to make a Web site by sketching out its appearance rather than its content, then you're going to have to unlearn some bad habits. As previously discussed, XML separates the content of a document from the appearance of the document. The content is developed first; then a format is attached to that content with a style sheet. Separating content from style is an extremely effective technique that improves both the content and the appearance of the document. Among other things, it allows authors and designers to work more independently of each other. However, it does require a different way of thinking about the design of a Web site, and perhaps even the use of different project-management techniques when multiple people are involved.

Cascading Style Sheets

Since XML allows arbitrary tags to be included in a document, there isn't any way for the browser to know in advance how each element should be displayed. When you send a document to a user you also need to send along a style sheet that tells the browser how to format individual elements. One kind of style sheet you can use is a Cascading Style Sheet (CSS).

CSS, initially designed for HTML, defines formatting properties like font size, font family, font weight, paragraph indentation, paragraph alignment, and other styles that can be applied to particular elements. For example, CSS allows HTML documents to specify that all `H1` elements should be formatted in 32 point centered Helvetica bold. Individual styles can be applied to most HTML tags that override the browser's defaults. Multiple style sheets can be applied to a single document, and multiple styles can be applied to a single element. The styles then cascade according to a particular set of rules.

CSS rules and properties are explored in more detail in Chapter 12, *Cascading Style Sheets Level 1*, and Chapter 13, *Cascading Style Sheets Level 2*.

It's easy to apply CSS rules to XML documents. You simply change the names of the tags you're applying the rules to. Mozilla 5.0 directly supports CSS style sheets combined with XML documents, though at present, it crashes rather too frequently.

Extensible Style Language

The Extensible Style Language (XSL) is a more advanced style-sheet language specifically designed for use with XML documents. XSL documents are themselves well-formed XML documents.

XSL documents contain a series of rules that apply to particular patterns of XML elements. An XSL processor reads an XML document and compares what it sees to the patterns in a style sheet. When a pattern from the XSL style sheet is recognized in the XML document, the rule outputs some combination of text. Unlike cascading style sheets, this output text is somewhat arbitrary and is not limited to the input text plus formatting information.

- CSS can only change the format of a particular element, and it can only do so on an element-wide basis. XSL style sheets, on the other hand, can rearrange and reorder elements. They can hide some elements and display others. Furthermore, they can choose the style to use not just based on the tag, but also on the contents and attributes of the tag, on the position of the tag in the document relative to other elements, and on a variety of other criteria.

- CSS has the advantage of broader browser support. However, XSL is far more flexible and powerful, and better suited to XML documents. Furthermore, XML documents with XSL style sheets can be easily converted to HTML documents with CSS style sheets.

XSL style sheets will be explored in great detail in Chapter 14, *XSL Transformations*, and Chapter 15, *XSL Formatting Objects*.

URLs and URIs

XML documents can live on the Web, just like HTML and other documents. When they do, they are referred to by Uniform Resource Locators (URLs), just like HTML files. For example, at the URL `http://www.hypermedic.com/style/xml/tempest.xml` you'll find the complete text of Shakespeare's *Tempest* marked up in XML.

Although URLs are well understood and well supported, the XML specification uses the more general Uniform Resource Identifier (URI). URIs are a more general architecture for locating resources on the Internet, that focus a little more on the resource and a little less on the location. In theory, a URI can find the closest copy of a mirrored document or locate a document that has been moved from one site to another. In practice, URIs are still an area of active research, and the only kinds of URIs that are actually supported by current software are URLs.

XLinks and XPointers

As long as XML documents are posted on the Internet, you're going to want to be able to address them and hot link between them. Standard HTML link tags can be used in XML documents, and HTML documents can link to XML documents. For example, this HTML link points to the aforementioned copy of the *Tempest* rendered in XML:

```
<a href="http://www.hypermedic.com/style/xml/tempest.xml">
  The Tempest by Shakespeare
</a>
```

Whether the browser can display this document if you follow the link, depends on just how well the browser handles XML files. Most current browsers don't handle them very well.

However, XML lets you go further with XLinks for linking to documents and XPointers for addressing individual parts of a document.

XLinks enable any element to become a link, not just an `A` element. Furthermore, links can be bi-directional, multidirectional, or even point to multiple mirror sites from which the nearest is selected. XLinks use normal URLs to identify the site they're linking to.

XLinks are discussed in Chapter 16, *XLinks*.

XPointers enable links to point not just to a particular document at a particular location, but to a particular part of a particular document. An XPointer can refer to a particular element of a document, to the first, the second, or the 17th such element, to the first element that's a child of a given element, and so on. XPointers provide extremely powerful connections between documents that do not require the targeted document to contain additional markup just so its individual pieces can be linked to it.

Furthermore, unlike HTML anchors, XPointers don't just refer to a point in a document. They can point to ranges or spans. Thus an XPointer might be used to select a particular part of a document, perhaps so that it can be copied or loaded into a program.

XPointers are discussed in Chapter 17, *XPointers*.

The Unicode Character Set

The Web is international, yet most of the text you'll find on it is in English. XML is starting to change that. XML provides full support for the two-byte Unicode character set, as well as its more compact representations. This character set supports almost every character commonly used in every modern script on Earth.

Unfortunately, XML alone is not enough. To read a script you need three things:

1. A character set for the script
2. A font for the character set
3. An operating system and application software that understands the character set

If you want to write in the script as well as read it, you'll also need an input method for the script. However, XML defines character references that allow you to use pure ASCII to encode characters not available in your native character set. This is sufficient for an occasional quote in Greek or Chinese, though you wouldn't want to rely on it to write a novel in another language.

In Chapter 7, *Foreign Languages and non-Roman Text*, you'll explore how international text is represented in computers, how XML understands text, and how you can use the software you have to read and write in languages other than English.

How the Technologies Fit Together

XML defines a grammar for tags you can use to mark up a document. An XML document is marked up with XML tags. The default encoding for XML documents is Unicode.

Among other things, an XML document may contain hypertext links to other documents and resources. These links are created according to the XLink specification. XLinks identify the documents they're linking to with URIs (in theory) or URLs (in practice). An XLink may further specify the individual part of a document it's linking to. These parts are addressed via XPointers.

If an XML document is intended to be read by human beings — and not all XML documents are — then a style sheet provides instructions about how individual elements are formatted. The style sheet may be written in any of several style-sheet languages. CSS and XSL are the two most popular style-sheet languages, though there are others including DSSSL — the Document Style Semantics and Specification Language — on which XSL is based.

I've outlined a lot of exciting stuff in this chapter. However, honesty compels me to tell you that I haven't discussed all of it yet. In fact, much of what I've described is the promise of XML rather than the current reality. XML has a lot of people in the software industry very excited, and a lot of programmers are working very hard to turn these dreams into reality. New software is released every day that brings us closer to XML nirvana, but this is all very new, and some of the software isn't fully cooked yet. Throughout the rest of this book, I'll be careful to point out not only what is supposed to happen, but what actually does happen. Depressingly these are all too often not the same thing. Nonetheless with a little caution you can do real work right now with XML.

Summary

In this chapter, you have learned some of the things that XML can do for you. In particular, you have learned:

- ✦ XML is a meta-markup language that enables the creation of markup languages for particular documents and domains.
- ✦ XML tags describe the structure and semantics of a document's content, not the format of the content. The format is described in a separate style sheet.
- ✦ XML grew out of many users' frustration with the complexity of SGML and the inadequacies of HTML.
- ✦ XML documents are created in an editor, read by a parser, and displayed by a browser.
- ✦ XML on the Web rests on the foundations provided by HTML, Cascading Style Sheets, and URLs.
- ✦ Numerous supporting technologies layer on top of XML, including XSL style sheets, XLinks, and XPointers. These let you do more than you can accomplish with just CSS and URLs.
- ✦ Be careful. XML isn't completely finished. It will change and expand, and you will encounter bugs in current XML software.

In the next chapter, you'll see a number of XML applications, and learn about some ways XML is being used in the real world today. Examples include vector graphics, music notation, mathematics, chemistry, human resources, Webcasting, and more.

✦ ✦ ✦

CHAPTER 5

Attributes, Empty Tags, and XSL

In This Chapter

Attributes

Attributes versus elements

Empty tags

XSL

You can encode a given set of data in XML in nearly an infinite number of ways. There's no one right way to do it although some ways are more right than others, and some are more appropriate for particular uses. In this chapter, we explore a different solution to the problem of marking up baseball statistics in XML, carrying over the baseball example from the previous chapter. Specifically, we will address the use of attributes to store information and empty tags to define element positions. In addition, since CSS doesn't work well with content-less XML elements of this form, we'll examine an alternative—and more powerful—style sheet language called XSL.

Attributes

In the last chapter, all data was categorized into the name of a tag or the contents of an element. This is a straightforward and easy-to-understand approach, but it's not the only one. As in HTML, XML elements may have attributes. An attribute is a name-value pair associated with an element. The name and the value are each strings, and no element may contain two attributes with the same name.

You're already familiar with attribute syntax from HTML. For example, consider this <IMG> tag:

```
<IMG SRC=cup.gif WIDTH=89 HEIGHT=67 ALT="Cup
of coffee">
```

It has four attributes, the SRC attribute whose value is cup.gif, the WIDTH attribute whose value is 89, the HEIGHT attribute whose value is 67, and the ALT attribute whose value is Cup of coffee. However, in XML-unlike HTML-attribute values must always be quoted and start tags must have matching close tags. Thus, the XML equivalent of this tag is:

```
<IMG SRC="cup.gif" WIDTH="89" HEIGHT="67" ALT="Cup of coffee">
</IMG>
```

Another difference between HTML and XML is that XML assigns no particular meaning to the IMG tag and its attributes. In particular, there's no guarantee that an XML browser will interpret this tag as an instruction to load and display the image in the file cup.gif.

You can apply attribute syntax to the baseball example quite easily. This has the advantage of making the markup somewhat more concise. For example, instead of containing a YEAR child element, the SEASON element only needs a YEAR attribute.

```
<SEASON YEAR="1998">
</SEASON>
```

On the other hand, LEAGUE should be a child of the SEASON element rather than an attribute. For one thing, there are two leagues in a season. Anytime there's likely to be more than one of something child elements are called for. Attribute names must be unique within an element. Thus you should not, for example, write a SEASON element like this:

```
<SEASON YEAR="1998" LEAGUE="National" League="American">
</SEASON>
```

The second reason LEAGUE is naturally a child element rather than an attribute is that it has substructure; it is subdivided into DIVISION elements. Attribute values are flat text. XML elements can conveniently encode structure-attribute values cannot.

However, the name of a league is unstructured, flat text; and there's only one name per league so LEAGUE elements can easily have a NAME attribute instead of a LEAGUE_NAME child element:

```
<LEAGUE NAME="National League">
</LEAGUE>
```

Since an attribute is more closely tied to its element than a child element is, you don't run into problems by using NAME instead of LEAGUE_NAME for the name of the attribute. Divisions and teams can also have NAME attributes without any fear of confusion with the name of a league. Since a tag can have more than one attribute (as long as the attributes have different names), you can make a team's city an attribute as well, as shown below:

```
<LEAGUE NAME="American League">
  <DIVISION NAME="East">
    <TEAM NAME="Orioles" CITY="Baltimore"></TEAM>
    <TEAM NAME="Red Sox" CITY="Boston"></TEAM>
    <TEAM NAME="Yankees" CITY="New York"></TEAM>
    <TEAM NAME="Devil Rays" CITY="Tampa Bay"></TEAM>
    <TEAM NAME="Blue Jays" CITY="Toronto"></TEAM>
  </DIVISION>
</LEAGUE>
```

Players will have a lot of attributes if you choose to make each statistic an attribute. For example, here are Joe Girardi's 1998 statistics as attributes:

```
<PLAYER GIVEN_NAME="Joe" SURNAME="Girardi"
  GAMES="78" AT_BATS="254" RUNS="31" HITS="70"
  DOUBLES="11" TRIPLES="4" HOME_RUNS="3"
  RUNS_BATTED_IN="31" WALKS="14" STRUCK_OUT="38"
  STOLEN_BASES="2" CAUGHT_STEALING="4"
  SACRIFICE_FLY="1" SACRIFICE_HIT="8"
  HIT_BY_PITCH="2">
</PLAYER>
```

Listing 5-1 uses this new attribute style for a complete XML document containing the baseball statistics for the 1998 major league season. It displays the same information (i.e., two leagues, six divisions, 30 teams, and nine players) as does Listing 4-1 in the last chapter. It is merely marked up differently. Figure 5-1 shows this document loaded into Internet Explorer 5.0 without a style sheet.

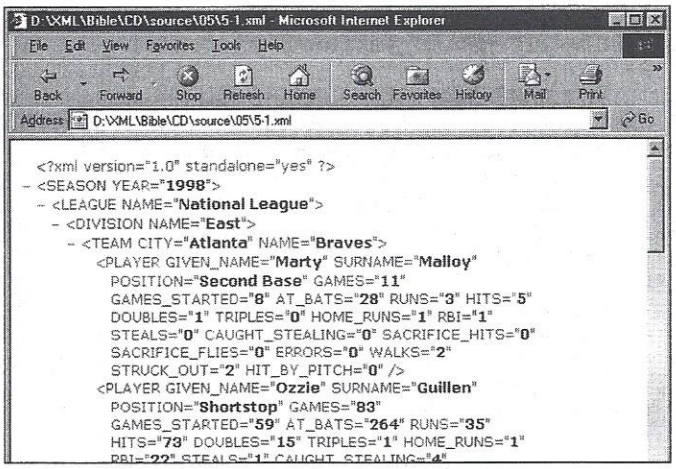


Figure 5-1: The 1998 major league baseball statistics using attributes for most information.

Listing 5-1: A complete XML document that uses attributes to store baseball statistics

```
<?xml version="1.0" standalone="yes"?>
<SEASON YEAR="1998">
  <LEAGUE NAME="National League">
    <DIVISION NAME="East">
      <TEAM CITY="Atlanta" NAME="Braves">
        <PLAYER GIVEN_NAME="Marty" SURNAME="Malloy"
         POSITION="Second Base" GAMES="11" GAMES_STARTED="8"
         AT_BATS="28" RUNS="3" HITS="5" DOUBLES="1"
         TRIPLES="0" HOME_RUNS="1" RBI="1" STEALS="0"
         CAUGHT_STEALING="0" SACRIFICE_HITS="0"
         SACRIFICE_FLIES="0" ERRORS="0" WALKS="2"
         STRUCK_OUT="2" HIT_BY_PITCH="0">
        </PLAYER>
        <PLAYER GIVEN_NAME="Ozzie" SURNAME="Guillen"
         POSITION="Shortstop" GAMES="83" GAMES_STARTED="59"
         AT_BATS="264" RUNS="35" HITS="73" DOUBLES="15"
         TRIPLES="1" HOME_RUNS="1" RBI="22" STEALS="1"
         CAUGHT_STEALING="4" SACRIFICE_HITS="4"
         SACRIFICE_FLIES="2" ERRORS="6" WALKS="24"
         STRUCK_OUT="25" HIT_BY_PITCH="1">
        </PLAYER>
        <PLAYER GIVEN_NAME="Danny" SURNAME="Bautista"
          POSITION="Outfield" GAMES="82" GAMES_STARTED="27"
          AT_BATS="144" RUNS="17" HITS="36" DOUBLES="11"
          TRIPLES="0" HOME_RUNS="3" RBI="17" STEALS="1"
          CAUGHT_STEALING="0" SACRIFICE_HITS="3"
          SACRIFICE_FLIES="2" ERRORS="2" WALKS="7"
          STRUCK_OUT="21" HIT_BY_PITCH="0">
        </PLAYER>
        <PLAYER GIVEN_NAME="Gerald" SURNAME="Williams"
         POSITION="Outfield" GAMES="129" GAMES_STARTED="51"
         AT_BATS="266" RUNS="46" HITS="81" DOUBLES="18"
         TRIPLES="3" HOME_RUNS="10" RBI="44" STEALS="11"
         CAUGHT_STEALING="5" SACRIFICE_HITS="2"
         SACRIFICE_FLIES="1" ERRORS="5" WALKS="17"
         STRUCK_OUT="48" HIT_BY_PITCH="3">
        </PLAYER>
        <PLAYER GIVEN_NAME="Tom" SURNAME="Glavine"
         POSITION="Starting Pitcher" GAMES="33"
         GAMES_STARTED="33" WINS="20" LOSSES="6" SAVES="0"
         COMPLETE_GAMES="4" SHUT_OUTS="3" ERA="2.47"
         INNINGS="229.1" HOME_RUNS_AGAINST="13"
         RUNS_AGAINST="67" EARNED_RUNS="63" HIT_BATTER="2"
         WILD_PITCHES="3" BALK="0" WALKED_BATTER="74"
         STRUCK_OUT_BATTER="157">
        </PLAYER>
        <PLAYER GIVEN_NAME="Javier" SURNAME="Lopez"
         POSITION="Catcher" GAMES="133" GAMES_STARTED="124"
         AT_BATS="489" RUNS="73" HITS="139" DOUBLES="21"
         TRIPLES="1" HOME_RUNS="34" RBI="106" STEALS="5"
```

```
      CAUGHT_STEALING="3" SACRIFICE_HITS="1"
      SACRIFICE_FLIES="8" ERRORS="5" WALKS="30"
      STRUCK_OUT="85" HIT_BY_PITCH="6">
    </PLAYER>
    <PLAYER GIVEN_NAME="Ryan" SURNAME="Klesko"
      POSITION="Outfield" GAMES="129" GAMES_STARTED="124"
      AT_BATS="427" RUNS="69" HITS="117" DOUBLES="29"
      TRIPLES="1" HOME_RUNS="18" RBI="70" STEALS="5"
      CAUGHT_STEALING="3" SACRIFICE_HITS="0"
      SACRIFICE_FLIES="4" ERRORS="2" WALKS="56"
      STRUCK_OUT="66" HIT_BY_PITCH="3">
    </PLAYER>
    <PLAYER GIVEN_NAME="Andres" SURNAME="Galarraga"
      POSITION="First Base" GAMES="153" GAMES_STARTED="151"
      AT_BATS="555" RUNS="103" HITS="169" DOUBLES="27"
      TRIPLES="1" HOME_RUNS="44" RBI="121" STEALS="7"
      CAUGHT_STEALING="6" SACRIFICE_HITS="0"
      SACRIFICE_FLIES="5" ERRORS="11" WALKS="63"
      STRUCK_OUT="146" HIT_BY_PITCH="25">
    </PLAYER>
    <PLAYER GIVEN_NAME="Wes" SURNAME="Helms"
      POSITION="Third Base" GAMES="7" GAMES_STARTED="2"
      AT_BATS="13" RUNS="2" HITS="4" DOUBLES="1"
      TRIPLES="0" HOME_RUNS="1" RBI="2" STEALS="0"
      CAUGHT_STEALING="0" SACRIFICE_HITS="0"
      SACRIFICE_FLIES="0" ERRORS="1" WALKS="0"
      STRUCK_OUT="4" HIT_BY_PITCH="0">
    </PLAYER>
  </TEAM>
  <TEAM CITY="Florida" NAME="Marlins">
  </TEAM>
  <TEAM CITY="Montreal" NAME="Expos">
  </TEAM>
  <TEAM CITY="New York" NAME="Mets">
  </TEAM>
  <TEAM CITY="Philadelphia" NAME="Phillies">
  </TEAM>
</DIVISION>
<DIVISION NAME="Central">
  <TEAM CITY="Chicago" NAME="Cubs">
  </TEAM>
  <TEAM CITY="Cincinnati" NAME="Reds">
  </TEAM>
  <TEAM CITY="Houston" NAME="Astros">
  </TEAM>
  <TEAM CITY="Milwaukee" NAME="Brewers">
  </TEAM>
  <TEAM CITY="Pittsburgh" NAME="Pirates">
  </TEAM>
  <TEAM CITY="St. Louis" NAME="Cardinals">
  </TEAM>
</DIVISION>
```

Continued

Listing 5-1 *(continued)*

```
      <DIVISION NAME="West">
        <TEAM CITY="Arizona" NAME="Diamondbacks">
        </TEAM>
        <TEAM CITY="Colorado" NAME="Rockies">
        </TEAM>
        <TEAM CITY="Los Angeles" NAME="Dodgers">
        </TEAM>
        <TEAM CITY="San Diego" NAME="Padres">
        </TEAM>
        <TEAM CITY="San Francisco" NAME="Giants">
        </TEAM>
      </DIVISION>
    </LEAGUE>
    <LEAGUE NAME="American League">
      <DIVISION NAME="East">
        <TEAM CITY="Baltimore" NAME="Orioles">
        </TEAM>
        <TEAM CITY="Boston" NAME="Red Sox">
        </TEAM>
        <TEAM CITY="New York" NAME="Yankees">
        </TEAM>
        <TEAM CITY="Tampa Bay" NAME="Devil Rays">
        </TEAM>
        <TEAM CITY="Toronto" NAME="Blue Jays">
        </TEAM>
      </DIVISION>
      <DIVISION NAME="Central">
        <TEAM CITY="Chicago" NAME="White Sox">
        </TEAM>
        <TEAM CITY="Kansas City" NAME="Royals">
        </TEAM>
        <TEAM CITY="Detroit" NAME="Tigers">
        </TEAM>
        <TEAM CITY="Cleveland" NAME="Indians">
        </TEAM>
        <TEAM CITY="Minnesota" NAME="Twins">
        </TEAM>
      </DIVISION>
      <DIVISION NAME="West">
        <TEAM CITY="Anaheim" NAME="Angels">
        </TEAM>
        <TEAM CITY="Oakland" NAME="Athletics">
        </TEAM>
        <TEAM CITY="Seattle" NAME="Mariners">
        </TEAM>
        <TEAM CITY="Texas" NAME="Rangers">
        </TEAM>
      </DIVISION>
    </LEAGUE>
  </SEASON>
```

Listing 5-1 uses only attributes for player information. Listing 4-1 used only element content. There are intermediate approaches as well. For example, you could make the player's name part of element content while leaving the rest of the statistics as attributes, like this:

```
<P>
  On Tuesday <PLAYER GAMES="78" AT_BATS="254" RUNS="31"
  HITS="70" DOUBLES="11" TRIPLES="4" HOME_RUNS="3"
  RUNS_BATTED_IN="31" WALKS="14" STRIKE_OUTS="38"
  STOLEN_BASES="2" CAUGHT_STEALING="4"
  SACRIFICE_FLY="1" SACRIFICE_HIT="8"
  HIT_BY_PITCH="2">Joe Girardi</PLAYER> struck out twice
  and...
</P>
```

This would include Joe Girardi's name in the text of a page while still making his statistics available to readers who want to look deeper, as a hypertext footnote or tool tip. There's always more than one way to encode the same data. Which way you pick generally depends on the needs of your specific application.

Attributes versus Elements

There are no hard and fast rules about when to use child elements and when to use attributes. Generally, you'll use whichever suits your application. With experience, you'll gain a feel for when attributes are easier than child elements and vice versa. Until then, one good rule of thumb is that the data itself should be stored in elements. Information about the data (meta-data) should be stored in attributes. And when in doubt, put the information in the elements.

To differentiate between data and meta-data, ask yourself whether someone reading the document would want to see a particular piece of information. If the answer is yes, then the information probably belongs in a child element. If the answer is no, then the information probably belongs in an attribute. If all tags were stripped from the document along with all the attributes, the basic information should still be present. Attributes are good places to put ID numbers, URLs, references, and other information not directly or immediately relevant to the reader. However, there are many exceptions to the basic principal of storing meta-data as attributes. These include:

- ✦ Attributes can't hold structure well.
- ✦ Elements allow you to include meta-meta-data (information about the information about the information).
- ✦ Not everyone always agrees on what is and isn't meta-data.
- ✦ Elements are more extensible in the face of future changes.

Structured Meta-data

One important principal to remember is that elements can have substructure and attributes can't. This makes elements far more flexible, and may convince you to encode meta-data as child elements. For example, suppose you're writing a paper and you want to include a source for a fact. It might look something like this:

```
<FACT SOURCE="The Biographical History of Baseball,
Donald Dewey and Nicholas Acocella (New York: Carroll &amp;amp;
Graf Publishers, Inc. 1995) p. 169">
  Josh Gibson is the only person in the history of baseball to
  hit a pitch out of Yankee Stadium.
</FACT>
```

Clearly the information "The Biographical History of Baseball, Donald Dewey and Nicholas Acocella (New York: Carroll & Graf Publishers, Inc. 1995) p. 169" is meta-data. It is not the fact itself. Rather it is information about the fact. However, the SOURCE attribute contains a lot of implicit substructure. You might find it more useful to organize the information like this:

```
<SOURCE>
  <AUTHOR>Donald Dewey</AUTHOR>
  <AUTHOR>Nicholas Acocella</AUTHOR>
  <BOOK>
    <TITLE>The Biographical History of Baseball</TITLE>
    <PAGES>169</PAGES>
    <YEAR>1995</YEAR>
  </BOOK>
</SOURCE>
```

Furthermore, using elements instead of attributes makes it straightforward to include additional information like the authors' e-mail addresses, a URL where an electronic copy of the document can be found, the title or theme of the particular issue of the journal, and anything else that seems important.

Dates are another common example. One common piece of meta-data about scholarly articles is the date the article was first received. This is important for establishing priority of discovery and invention. It's easy to include a DATE attribute in an ARTICLE tag like this:

```
<ARTICLE DATE="06/28/1969">
  Polymerase Reactions in Organic Compounds
</ARTICLE>
```

However, the DATE attribute has substructure signified by the /. Getting that structure out of the attribute value, however, is much more difficult than reading child elements of a DATE element, as shown below:

```
<DATE>
  <YEAR>1969</YEAR>
  <MONTH>06</MONTH>
  <DAY>28</DAY>
</DATE>
```

For instance, with CSS or XSL, it's easy to format the day and month invisibly so that only the year appears. For example, using CSS:

```
YEAR {display: inline}
MONTH {display: none}
DAY {display: none}
```

If the DATE is stored as an attribute, however, there's no easy way to access only part of it. You must write a separate program in a programming language like ECMAScript or Java that can parse your date format. It's easier to use the standard XML tools and child elements.

Furthermore, the attribute syntax is ambiguous. What does the date "10/11/1999" signify? In particular, is it October 11th or November 10th? Readers from different countries will interpret this data differently. Even if your parser understands one format, there's no guarantee the people entering the data will enter it correctly. The XML, by contrast, is unambiguous.

Finally, using DATE children rather than attributes allows more than one date to be associated with an element. For instance, scholarly articles are often returned to the author for revisions. In these cases, it can also be important to note when the revised article was received. For example:

```
<ARTICLE>
  <TITLE>
    Maximum Projectile Velocity in an Augmented Railgun
  </TITLE>
  <AUTHOR>Elliotte Harold</AUTHOR>
  <AUTHOR>Bruce Bukiet</AUTHOR>
  <AUTHOR>William Peter</AUTHOR>
  <DATE>
    <YEAR>1992</YEAR>
    <MONTH>10</MONTH>
    <DAY>29</DAY>
  </DATE>
  <DATE>
    <YEAR>1993</YEAR>
    <MONTH>10</MONTH>
    <DAY>26</DAY>
  </DATE>
</ARTICLE>
```

As another example, consider the ALT attribute of an IMG tag in HTML. This is limited to a single string of text. However, given that a picture is worth a thousand words, you might well want to replace an IMG with marked up text. For instance, consider the pie chart shown in Figure 5-2.

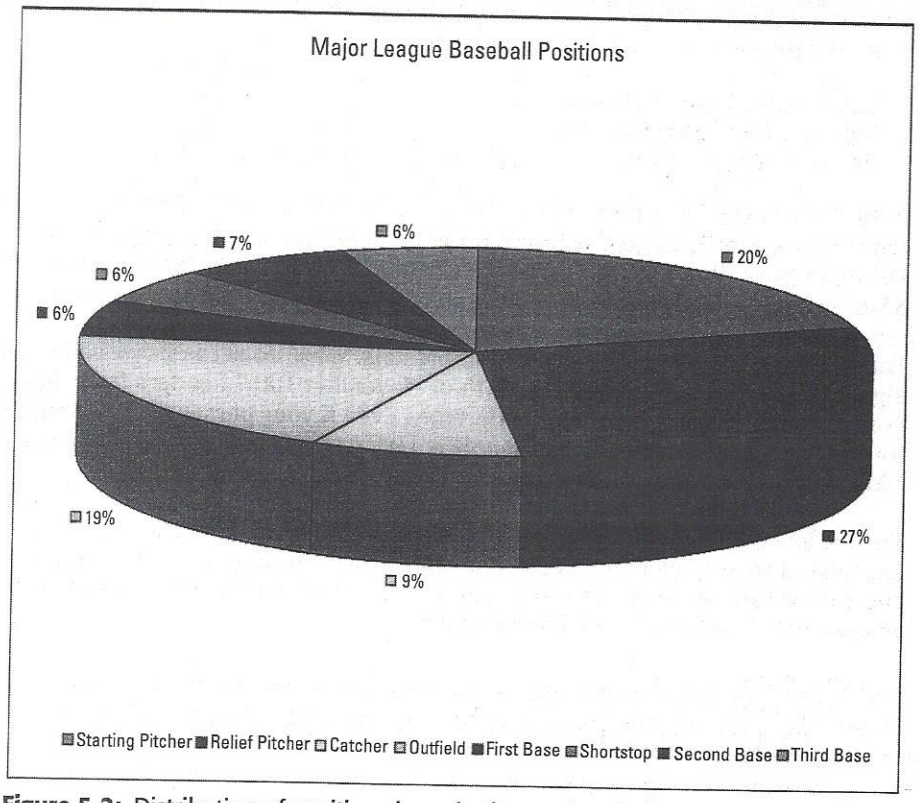


Figure 5-2: Distribution of positions in major league baseball

Using an ALT attribute, the best description of this picture you can provide is:

```
<IMG SRC="05021.gif"
     ALT="Pie Chart of Positions in Major League Baseball"
     WIDTH="819" HEIGHT="623">
</IMG>
```

However, with an ALT child element, you have more flexibility because you can embed markup. For example, you might provide a table of the relevant numbers instead of a pie chart.

```
<IMG SRC="05021.gif" WIDTH="819" HEIGHT="623">
  <ALT>
    <TABLE>
      <TR>
        <TD>Starting Pitcher</TD> <TD>242</TD> <TD>20%</TD>
      </TR>
      <TR>
        <TD>Relief Pitcher</TD> <TD>336</TD> <TD>27%</TD>
      </TR>
      <TR>
        <TD>Catcher</TD> <TD>104</TD> <TD>9%</TD>
      </TR>
      <TR>
        <TD>Outfield</TD> <TD>235</TD> <TD>19%</TD>
      </TR>
      <TR>
        <TD>First Base</TD> <TD>67</TD> <TD>6%</TD>
      </TR>
      <TR>
        <TD>Shortstop</TD> <TD>67</TD> <TD>6%</TD>
      </TR>
      <TR>
        <TD>Second Base</TD> <TD>88</TD> <TD>7%</TD>
      </TR>
      <TR>
        <TD>Third Base</TD> <TD>67</TD> <TD>6%</TD>
      </TR>
    </TABLE>
  </ALT>
</IMG>
```

You might even provide the actual Postscript, SVG, or VML code to render the picture in the event that the bitmap image is not available.

Meta-Meta-Data

Using elements for meta-data also easily allows for meta-meta-data, or information about the information about the information. For example, the author of a poem may be considered to be meta-data about the poem. The language in which that author's name is written is data about the meta-data about the poem. This isn't a trivial concern, especially for distinctly non-Roman languages. For instance, is the author of the Odyssey Homer or Ωμηος? If you use elements, it's easy to write:

```
<POET LANGUAGE="English">Homer</POET>
<POET LANGUAGE="Greek">Ωμηος</POET>
```