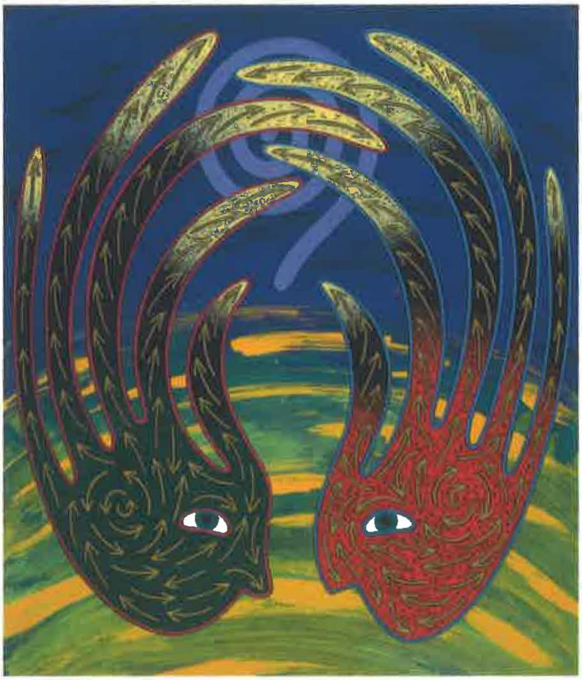
Merrill Communications LLC
d/b/a Merrill Corporation
Exhibit 1006 pt. 1

THE XML HANDBOOK™



- The definitive XML resource: applications, products, and technologies!
- Leverage your Web and intranet expertise with structured information
- CD-ROM: Unique trialware, demos, examples, specs, and more—plus 55 great, genuinely free XML software packages

CHARLES F. GOLDFARB · PAUL PRESCOD

THE DEFINITIVE XML SERIES FROM CHARLES F. GOLDFARB

The XML Handbook™

ISBN 0-13-081152-1

90000

9 780130 811523

The Charles F. Goldfarb Series on Open Information Management

"Open Information Management" (OIM) means managing information so that it is open to processing by any program, not just the program that created it. That extends even to application programs not conceived of at the time the information was created.

OIM is based on the principle of data independence: data should be stored in computers in non-proprietary, genuinely standardized representations. And that applies even when the data is the content of a document. Its representation should distinguish the innate information from the proprietary codes of document processing programs and the artifacts of particular presentation styles.

Business data bases—which rigorously separate the real data from the input forms and output reports—achieved data independence decades ago. But documents, unlike business data, have historically been created in the context of a particular output presentation style. So for document data, independence was largely unachievable until recently.

That is doubly unfortunate. It is unfortunate because documents are a far more significant repository of humanity's information. And documents can contain significantly richer information structures than data bases.

It is also unfortunate because the need for OIM of documents is greater now than ever. The demands of "repurposing" require that information be deliverable in multiple formats: paper-based, online, multimedia, hypermedia. And information must now be delivered through multiple channels: traditional bookstores and libraries, the World Wide Web, corporate intranets and extranets. In the latter modes, what starts as data base data may become a document for browsing, but then may need to be reused by the reader as data.

Fortunately, in the past ten years a technology has emerged that extends to documents the data base's capacity for data independence. And it does so without the data base's restrictions on structural freedom. That technology is the "Standard Generalized Markup Language" (SGML), an official International Standard (ISO 8879) that has been adopted by the world's largest producers of documents and by the World Wide Web.

With SGML, organizations in government, aerospace, airlines, automotive, electronics, computers, and publishing (to name a few) have freed their documents from hostage relationships to processing software. SGML coexists with graphics, multimedia and other data standards needed for OIM and acts as the framework that relates objects in the other formats to one another and to SGML documents.

The World Wide Web's HTML and XML are both based on SGML. HTML is a particular, though very general, application of SGML, like those for the above industries. There is a limited set of markup tags that can be used with HTML. XML, in contrast, is a simplified subset of SGML facilities that, like full SGML, can be used with any set of tags. You can literally create your own markup language with XML.

As the enabling standard for OIM of documents, the SGML family of standards necessarily plays a leading role in this series. We provide tutorials on SGML, XML, and other key standards and the techniques for applying them. Our books vary in technical intensity from programming techniques for software developers to the business justification of OIM for enterprise executives. We share the practical experience of organizations and individuals who have applied the techniques of OIM in environments ranging from immense industrial publishing projects to websites of all sizes.

Our authors are expert practitioners in their subject matter, not writers hired to cover a "hot" topic. They bring insight and understanding that can only come from real-world experience. Moreover, they practice what they preach about standardization. Their books share a common standards-based vocabulary. In this way, knowledge gained from one book in the series is directly applicable when reading another, or the standards themselves. This is just one of the ways in which we strive for the utmost technical accuracy and consistency with the OIM standards.

And we also strive for a sense of excitement and fun. After all, the challenge of OIM—preserving information from the ravages of technology while exploiting its benefits—is one of the great intellectual adventures of our age. I'm sure you'll find this series to be a knowledgable and reliable guide on that adventure.

About the Series Editor

Dr. Charles F. Goldfarb invented the SGML language in 1974 and later led the team that developed it into the International Standard on which both HTML and XML are based. He serves as editor of the Standard (ISO 8879) and as a consultant to developers of SGML and XML applications and products. He is based in Saratoga, CA.

About the Series Logo

The rebus is an ancient literary tradition, dating from 16th century Picardy, and is especially appropriate to a series involving fine distinctions between things and the words that describe them. For the logo, Andrew Goldfarb incorporated a rebus of the series name within a stylized SGML/XML comment declaration.

The Charles F. Goldfarb Series on Open Information Management

As XML is a subset of SGML, the Series List is categorized to show the degree to which a title applies to XML. "XML Titles" are those that discuss XML explicitly and may also cover full SGML. "SGML Titles" do not mention XML per se, but the principles covered may apply to XML.

XML Titles

Goldfarb, Pepper, and Ensign

- SGML Buyer's Guide™: Choosing the Right XML and SGML Products and Services

Megginson

- Structuring XML Documents

Leventhal, Lewis, and Fuchs

- Designing XML Internet Applications

Goldfarb and Prescod

- The XML Handbook™

Jelliffe

- The XML and SGML Cookbook: Recipes for Structured Information

SGML Titles

Turner, Douglass, and Turner

- ReadMe.1st: SGML for Writers and Editors

Donovan

- Industrial-Strength SGML: An Introduction to Enterprise Publishing

Ensign

- SGML: The Billion Dollar Secret

Rubinsky and Maloney

- SGML on the Web: Small Steps Beyond HTML

McGrath

- ParseMe.1st: SGML for Software Developers

DuCharme

- SGML CD

The XML Handbook™

Charles F. Goldfarb

Paul Prescod

Prentice Hall PTR, Upper Saddle River, NJ 07458
http://www.phptr.com

Library of Congress Cataloging-in-Publication Data

Goldfarb, Charles F.
XML handbook / Charles F. Goldfarb, Paul Prescod.
p. cm. -- (Charles F. Goldfarb series on open information management)
Includes index.
ISBN 0-13-081152-1 (pbk. : alk. paper)
1. XML (Document markup language) I. Prescod, Paul. II. Title. III. Series.
QA76.76.H92G65 1998
005.7'2--dc21 98-16708
CIP

Editorial/Production Supervision: *Patti Guerrieri*
Acquisitions Editor: *Mark L. Taub*
Editorial Assistant: *Audri Bazlan*
Marketing Manager: *Dan Rush*
Manufacturing Manager: *Alexis R. Heydt*
Cover Design: *Anthony Gemmellaro*
Cover Design Direction: *Jerry Votta*
Series Design: *Gail Cocker-Bogusz*

Prentice-Hall, Inc.
A Simon & Schuster Company
Upper Saddle River, NJ 07458

Printed in the United States of America

10 9 8 7 6 5 4 3 2

ISBN 0-13-081152-1

Prentice-Hall International (UK) Limited, London
Prentice-Hall of Australia Pty. Limited, Sydney
Prentice-Hall Canada Inc., Toronto
Prentice-Hall Hispanoamericana, S.A., Mexico
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Simon & Schuster Asia Pte. Ltd., Singapore
Editora Prentice-Hall do Brasil, Ltda., Rio de Janeiro

ISO 8879:1986, Information processing — Text and office systems — Standard Generalized Markup Language (SGML).

Complete copies of this standard can be obtained from the national member body of ISO in your country, or contact ISO, case postale 56, CH-1211 Geneva 20, Switzerland.

Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/REC-xml, W3C Recommendation 10-February-1998.

Extensible Linking Language (XLink), http://www.w3.org/TR/WD-xlink, W3C Working Draft 3-March-1998.

XML Pointer Language (XPointer), http://www.w3.org/TR/WD-xptr, W3C Working Draft 03-March-1998.

A Proposal for XSL, http://www.w3.org/TR/NOTE-XSL-970910, Note submitted to W3C on 27 August 1997.

HTML 4.0 Specification, http://www.w3.org/TR/REC-html40/, W3C Recommendation, revised on 24-Apr-1998.

The development of this book was partly subsidized by Sponsors, who provided both financial support and expert assistance in preparing the initial draft of the text identified with their names. However, as the Authors exercised final editorial control over the book, the Sponsors are in no way responsible for its content. In particular, opinions expressed in this book are those of the Authors and are not necessarily those of the Series Editor, Sponsors, or Publisher.

This book, and the CD-ROM included with it, contain software and descriptive materials provided by (or adapted from materials made publicly available by) product developers, vendors, and service providers. Said software and materials have not been reviewed, edited, or tested, and neither the Authors, Contributors, Series Editor, Publisher, Sponsors, or other parties connected with this book are responsible for their accuracy or reliability. Readers are warned that they use said software and materials at their own risk, and are urged to test the software and confirm the validity of the information prior to use.

To Linda – With love, awe, and gratitude.

Charles F. Goldfarb

For Lilia – Your support makes it possible and
your love makes it worthwhile.

Paul Prescod

Contents

Foreword

XML Everywhere

When HTML came onto the scene it sparked a publishing phenomenon. Ordinary people everywhere began to publish documents on the Web. Presentation on the Web became a topic of conversation not just within the computer industry, but within coffeehouses. Overnight, it seemed as though everyone had a Web page.

I see the same phenomenon happening today with XML. Where data was once a mysterious binary blob, it has now become something ordinary people can read and author because it's text. With XML, ordinary people have the ability to craft their own data, the ability to shape and control data. The significance of this shift is difficult to overstate, for not only does it mean that more people can access data, but that there will undoubtedly be more data to access. We are on the verge of a data explosion. One ignited by XML.

By infusing the Web with data, XML makes the Web a better place for people to interact, to do business. XML allows us to do more precise searches, deliver software components, describe such things as collections of Web pages and electronic commerce transactions, and much more. XML is

We also thank Lilia Prescod, Thea Prescod, and Linda Goldfarb for serving as our useability test laboratory. That means they read lots of chapters and complained until we made them clear enough.

Prentice Hall PTR uses Adobe FrameMaker to compose the books in my series. We thank Lani Hajagos of Adobe for providing Paul and me with copies.

Paul and I designed, and Paul implemented, an SGML-based production system for the book. It uses James Clark's Jade DSSSL processor, FrameMaker+SGML, and some ingenious FrameMaker plug-ins designed and implemented by Doug Yagaloff of Caxton, Inc. We thank Doug, and also Randy Kelley, for their wizard-level FrameMaker consulting advice.

But a great production system is nothing without a great Production Editor. We were fortunate to have Patti Guerrieri, who epitomizes grace – and skill – under fire. She coped with an untested system and a book that doubled in size, and still met the deadline.

This was my second project in which Linda Burman served as marketing consultant. I thank her – again – for her sage counsel and always cheerful encouragement.

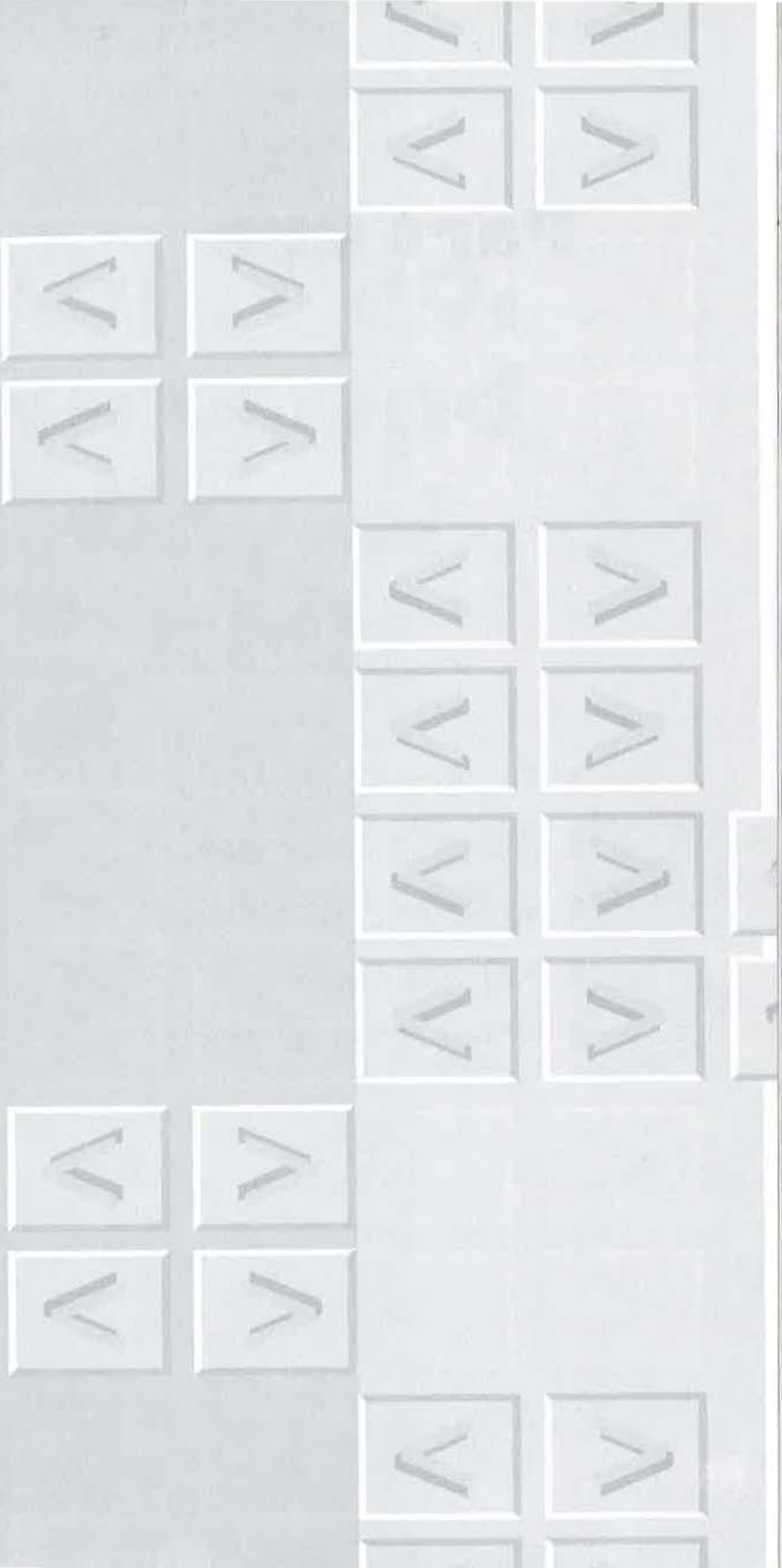
My personal thanks, also, to Mark Taub, now an Editor-in-Chief at Prentice Hall PTR, for his help, encouragement, and management of the project.

As the senior author, I gave myself the preface to write. I'm senior because Paul's folks were conceiving him about the same time that I was conceiving SGML. (In return, Paul got to write the history chapter, because for him it really is history.)

This gives me the opportunity to thank Paul publicly for the tremendous reservoir of talent, energy, and good humor that he brought to the project. The book benefitted not just from his XML knowledge and fine writing skills, but from his expertise in SGML, Jade, and FrameMaker that enabled us to automate the production of the book (with the previously acknowledged help from our friends).

Thanks, Paul.

Charles F. Goldfarb
Saratoga, CA
May 15, 1998



Foreword

XML Everywhere

When HTML came onto the scene it sparked a publishing phenomenon. Ordinary people everywhere began to publish documents on the Web. Presentation on the Web became a topic of conversation not just within the computer industry, but within coffeehouses. Overnight, it seemed as though everyone had a Web page.

I see the same phenomenon happening today with XML. Where data was once a mysterious binary blob, it has now become something ordinary people can read and author because it's text. With XML, ordinary people have the ability to craft their own data, the ability to shape and control data. The significance of this shift is difficult to overstate, for not only does it mean that more people can access data, but that there will undoubtedly be more data to access. We are on the verge of a data explosion. One ignited by XML.

By infusing the Web with data, XML makes the Web a better place for people to interact, to do business. XML allows us to do more precise searches, deliver software components, describe such things as collections of Web pages and electronic commerce transactions, and much more. XML is changing not only the way we think about data, but the way we think about the Web.

And by doing so, it's changing the way we think about the traditional desktop application. I have already witnessed the impact of XML on all types of applications from word processors and spreadsheets to database managers and email. More and more, such applications are reaching out to the Web, tapping into the power of the Web, and it is XML that is enabling them to do so. Gone are the days of the isolated, incompatible application. Here are the days of universal access and shared data.

I joined Microsoft in the summer of 1996 with great faith in the Standard Generalized Markup Language (SGML) and a dream that its potential might one day be realized. As soon as I arrived at Microsoft, Jon Bosak of Sun Microsystems and I began discussing the possibility of creating an XML standard. Jon shared my enthusiasm for a markup language such as XML, understanding what it could mean to Web communication.

My goal in designing an XML standard was to produce a very simple markup language with as few abstractions as possible. Microsoft's success is due in no small part to its ability to develop products with mass-market appeal. It is this mass-market appeal that I wanted to bring to XML. Together with Jon and other long-time friends from the SGML world, C.M. Sperberg-McQueen, James Clark, Tim Bray, Steve DeRose, Eve Maler, Eliot Kimber, Dave Hollander, Makoto Murata, and Peter Sharpe, I co-designed the XML specification at the World Wide Web Consortium (W3C). This specification, I believe, reflects my original goals.

It was truly an exciting time. For years, we had all been part of a maverick band of text markup enthusiasts, singing its praises every chance we had, and before us was an opportunity to bring XML into the mainstream, maybe even into the operating system. At last, we were getting our chance to tell the World of the thing we had been so crazy about for all this time.

By the fall of 1996, many groups inside Microsoft, including Office, the Site Server Electronic Commerce Edition, the Data Access Group, to cite a few, were searching for an open format to enable interoperability on the Web. It was then that I began working with the managers of Internet Explorer 4, with the passionate Adam Bosworth, with Andrew Layman, with Thomas Reardon, to define the Channel Definition Format (CDF). CDF, the first major application of XML on the Web, became an immediate and incredible success, and XML started catching on like wildfire across the Web.

I remember those weeks and months that followed as a time where it seemed that everyday another new group within Microsoft began coding applications using XML. Developers, left and right, were turning on to XML. They frenetically began to develop applications using XML, because XML gave them what they wanted: an easy-to-parse syntax for representing data. This flurry of activity was so great that by October of 1997, almost a year after my arrival at Microsoft, Chairman Bill Gates announced XML as "a breakthrough technology." Since that time we've never looked back.

This book is an excellent starting point where you can learn and experiment with XML. As the inventor of SGML, Dr. Charles F. Goldfarb is one of the most respected authorities on structured information. Charles has had a very direct influence on XML, as XML is a true subset of SGML, and he clearly understands the impact that XML will have on the world of data-driven, Web-based applications.

Charles and I share a common vision, that the most valuable asset for the user or for a corporation, namely the data, can be openly represented in a simple, flexible, and human-readable form. That it can easily travel from server to server, from server to client, and from application to application, fostering universal communication with anyone, anywhere. This vision can now be realized through XML.

Enjoy the book!

Redmond, April 24, 1988

Jean Paoli
Product Unit Manager, XML Technologies
Microsoft Corporation
Co-editor of the XML Specification

Preface

The World Wide Web is undergoing a radical change that will introduce wonderful services for users and amazing new opportunities for Web site developers and businesses.

HTML – the HyperText Markup Language – made the Web the world's library. Now its sibling, XML – the Extensible Markup Language – has begun to make the Web the world's commercial and financial hub. XML has just been approved as a W3C Recommendation, and already there are millions of XML files out there, with more coming online every day.

You can see why by comparing XML and HTML. Both are based on SGML – the International Standard for structured information – but look at the difference:

In HTML:

```
<p>P200 Laptop
<br>Friendly Computer Shop
<br>$1438
```

In XML:

```
<product>
<model>P200 Laptop</model>
<dealer>Friendly Computer Shop</dealer>
<price>$1438</price>
</product>
```

Both of these may appear the same in your browser, but the XML data is *smart* data. HTML tells how the data should *look*, but XML tells you what it *means.*

With XML, your browser knows there is a product, and it knows the model, dealer, and price. From a group of these it can show you the cheapest product or closest dealer without going back to the server.

Unlike HTML, with XML you create your own tags, so they describe exactly what you need to know. Because of that, your client-side applications can access data sources anywhere on the Web, in any format. New "middle-tier" servers sit between the data sources and the client, translating everything into your own task-specific XML.

But XML data isn't just smart data, it's also a smart document. That means when you display the information, the model name can be a different font from the dealer name, and the lowest price can be highlighted in green. Unlike HTML, where text is just text to be rendered in a uniform way, with XML text is smart, so it can control the rendition.

And you don't have to decide whether your information is data or documents; in XML, it is always both at once. You can do data processing or document processing or both at the same time.

With that kind of flexibility, it's no wonder that we're starting to see a Brave New Web of smart, structured information. Your broker sends your account data to Quicken using XML. Your "push" technology channel definitions are in XML. Everything from math to multimedia, chemistry to CommerceNet, is using XML or is preparing to start.

You should be too!

Welcome to the Brave New XML Web.

What about SGML?

This book is about XML. You won't find feature comparisons to SGML, or footnotes with nerdy observations like "the XML empty-element tag does not contradict the rule that every element has a start-tag and an end-tag because, in SGML terms, it is actually a start-tag followed immediately by a null end-tag".[1]

Nevertheless, for readers who use SGML, it is worth addressing the question of how XML and SGML relate. There has been a lot of speculation about this.

[1] Well, yes, I did just make that nerdy observation, but it wasn't a footnote, was it?

Some claim that XML will replace SGML because there will be so much free and low-cost software. Others assert that XML users, like HTML users before them, will discover that they need more of SGML and will eventually migrate to the full standard.

Both assertions are nonsense ... XML and SGML don't even compete.

XML is a simplified subset of SGML. The subsetting was optimized for the Web environment, which implies data-processing-oriented (rather than publishing-oriented), short life-span (in fact, usually dynamically-generated) information. The vast majority of XML documents will be created by computer programs and processed by other programs, then destroyed. Humans will never see them.

Eliot Kimber, a member of both the XML and SGML standards committees, says:

> There are certain use domains for which XML is simply not sufficient and where you need the additional features of SGML. These applications tend to be very large scale and of long term; e.g., aircraft maintenance information, government regulations, power plant documentation, etc.
>
> Any one of them might involve a larger volume of information than the entire use of XML on the Web. A single model of commercial aircraft, for example, requires some four million unique pages of documentation that must be revised and republished quarterly. Multiply that by the number of models produced by companies like Airbus and Boeing and you get a feel for the scale involved.

I invented SGML, I'm proud of it, and I'm awed that such a staggering volume of the world's mission-critical information is represented in it.

I'm also proud of XML. I'm proud of my friend Jon Bosak who made it happen, and I'm excited that the World Wide Web is becoming XML-based.

If you are new to XML, don't worry about any of this. All you need to know is that the XML subset of SGML has been in use for a decade or more, so you can trust it.

I am writing this the day after a meeting of the ISO committee that develops the SGML standard. We had the largest attendance in our 20-year history at that meeting. Interest in SGML has never been higher.

You should share that interest if you produce documents on the scale of an Airbus or Boeing. For the rest of us, there's XML.

About our sponsors

With all the buzz surrounding a hot technology like XML, it can be tough for a newcomer to distinguish the solid projects and realistic applications from the fluff and the fantasies. Our solution was to seek out companies with real products and realistic applications and tell their stories in sufficient detail that readers can see for themselves what is believable.

The application chapters are about what can be done with XML, extrapolating from actual experience with one or more users or prototype implementations. The case studies describe the XML experiences of specific named enterprises.

Some applications and case studies were done with full SGML before XML had a formal existence, but are within XML's capabilities. These are described as having been done with XML. Part of the proof of XML's viability is that people have used its core functions for over a decade.

The primary purpose of the tool chapters is to provide the vicarious experience of using a variety of XML tools without the effort of obtaining evaluation copies and installing them. They also provide useful information about uses and benefits of XML in general, which supplements the application-oriented discussions in the earlier parts of the book.

There are also two sponsored chapters on new XML-related technologies.

All sponsored chapters are identified with the name of the sponsor, and sometimes with the names of the experts who prepared the original text. All of the chapters were edited by me, sometimes extensively, in order to integrate them into the book. The editing objectives were to establish consistency of terminology and style, and to eliminate unnecessary duplication among the chapters. I believe the result was faithful to the intentions of the expert preparers with regard to bringing out the important characteristics of their applications and products.

The sponsorship program was organized by Linda Burman, the president of L. A. Burman Associates, a consulting company that provides marketing and business development services to the XML and SGML industries.

We are grateful to our sponsors just as we are grateful to you, our readers. Both of you together make it possible for the *XML Handbook* to exist. In the interests of everyone, we make our own editorial decisions and we don't recommend or endorse any product or service offerings over any others.

Our fourteen sponsors are:

- Adobe Systems Incorporated, http://www.adobe.com
- ArborText, Inc., http://www.arbortext.com
- Chrystal Software, http://www.chrystal.com
- Frank Russell Company Advanced Technology Labs, http://www.russell.com
- Inso Corporation, http://www.inso.com
- Interleaf, Inc., http://www.interleaf.com
- ISOGEN International, http://www.isogen.com
- Junglee Corporation, http://www.junglee.com
- Microsoft Corporation, http://www.microsoft.com
- Microstar Software Ltd., http://www.microstar.com
- POET Corporation, http://www.poet.com
- SoftQuad Inc., http://www.sq.com
- Texcel International, http://www.texcel.com
- webMethods, http://www.webmethods.com

Acknowledgments

The principal acknowledgment in a book of this nature has to be to the people who created the subject matter. In this case, I take special pleasure in the fact that all of them are friends and colleagues of long standing in the SGML community.

Tim Bray and C. Michael Sperberg-McQueen were the original editors of the XML specification, later joined by Jean Paoli. Dan Connolly put the project on the W3C "todo list" and shepherded it through the approval process.

But all of them agree that, if a single person is to be thanked for XML, it is Jon Bosak. Jon not only sparked the original ideas and recruited the team, but organized and chairs the W3C XML Working Group.

As Tim put it: "Without Jon, XML wouldn't have happened. He was the prime mover."

Regarding the content of the book, Paul and I would like to thank Jean Paoli, Eliot Kimber, David Siegel, Andy Goldfarb, Lars Marius Garshol, and Steve Newcomb for contributing great material; Bryan Bell, inventor of MIDI and document system architect extraordinaire, for his advice and support; Steve Pepper and Bob DuCharme for talent-spotting and Richard Lander for his insights into XSL.

We also thank Lilia Prescod, Thea Prescod, and Linda Goldfarb for serving as our useability test laboratory. That means they read lots of chapters and complained until we made them clear enough.

Prentice Hall PTR uses Adobe FrameMaker to compose the books in my series. We thank Lani Hajagos of Adobe for providing Paul and me with copies.

Paul and I designed, and Paul implemented, an SGML-based production system for the book. It uses James Clark's Jade DSSSL processor, FrameMaker+SGML, and some ingenious FrameMaker plug-ins designed and implemented by Doug Yagaloff of Caxton, Inc. We thank Doug, and also Randy Kelley, for their wizard-level FrameMaker consulting advice.

But a great production system is nothing without a great Production Editor. We were fortunate to have Patti Guerrieri, who epitomizes grace – and skill – under fire. She coped with an untested system and a book that doubled in size, and still met the deadline.

This was my second project in which Linda Burman served as marketing consultant. I thank her – again – for her sage counsel and always cheerful encouragement.

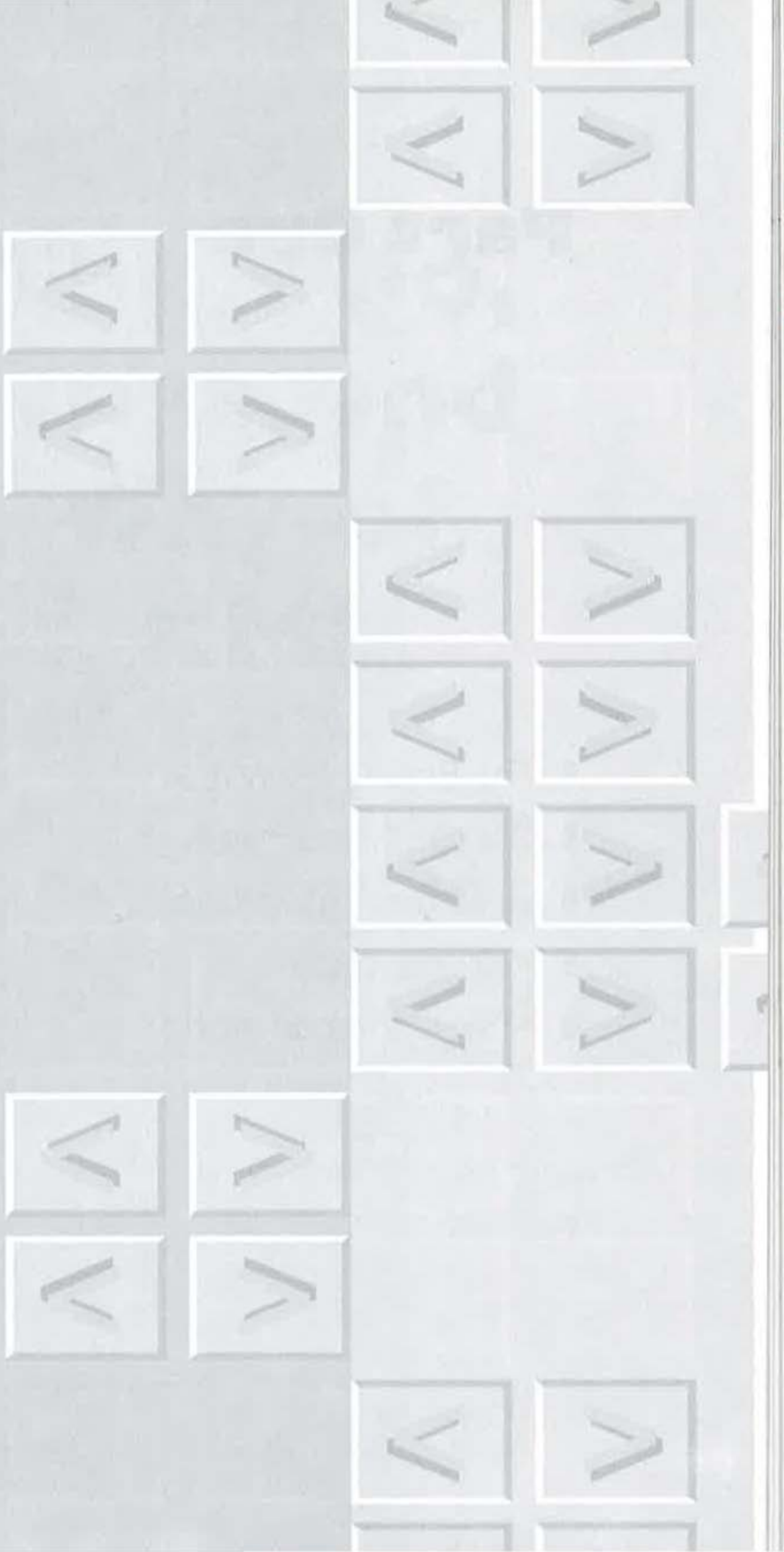
My personal thanks, also, to Mark Taub, now an Editor-in-Chief at Prentice Hall PTR, for his help, encouragement, and management of the project.

As the senior author, I gave myself the preface to write. I'm senior because Paul's folks were conceiving him about the same time that I was conceiving SGML. (In return, Paul got to write the history chapter, because for him it really is history.)

This gives me the opportunity to thank Paul publicly for the tremendous reservoir of talent, energy, and good humor that he brought to the project. The book benefitted not just from his XML knowledge and fine writing skills, but from his expertise in SGML, Jade, and FrameMaker that enabled us to automate the production of the book (with the previously acknowledged help from our friends).

Thanks, Paul.

Charles F. Goldfarb
Saratoga, CA
May 15, 1998



Part One

- The Brave New Web
- Data and documents
- Structured information
- XML concepts
- XML in the real world

The Who, What, and Why of XML

Why XML?

- What is XML, really?
- Origins in document processing
- Abstraction vs. rendition
- Documents and data

Chapter 1

Many of the most influential companies in the software industry are promoting XML as the next step in the Web's evolution. How can they be so confident about something so new? More important: how can *you* be sure that your time invested in learning and using XML will be profitable?

We can all safely bet on XML because the central ideas in this new technology are in fact very old and have been proven correct across several decades and thousands of projects. The easiest way to understand these ideas is to go back to their source, the *Standard Generalized Markup Language* (SGML).

XML is, in fact, a streamlined subset of SGML, so SGML's track record is XML's as well.

And if your interest is in moving data from Web sites to a browser or a spreadsheet, stay with us. All of this is interconnected and extremely relevant. For the amazing truth about XML is that with it, data processing and document processing are the same thing! If you understand where it all comes from, you'll understand where it – and the Web – are going.

1.1 | Text formatters and SGML

XML comes from a rich history of text processing systems. *Text processing* is the subdiscipline of *computer science* dedicated to creating computer systems that can automate parts of the document creation and publishing process. Text processing software includes simple word processors, advanced news item databases, hypertext document presentation systems and other publishing tools.

The first wave of automated text processing was computer typesetting. Authors would type in a document and describe how they would like it to be formatted. The computer would print out a document with the described text and formatting.

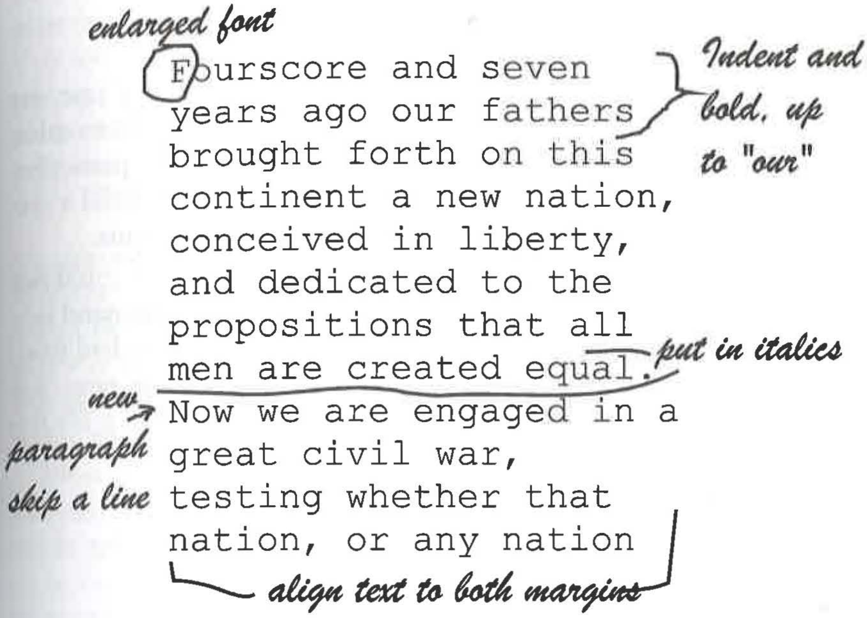
We call the file format that contained the mix of the actual data of the document, plus the description of the desired format, a *rendition*. Some well-known rendition notations include *troff*, *Rich Text Format* (*RTF*), and *LaTeX*.

The system would convert the rendition into something physically perceivable to a human being – a *presentation*. The presentation medium was historically paper, but eventually electronic display.

Typesetting systems sped up the process of publishing documents and evolved into what we now know as desktop publishing. Newer programs like *Microsoft Word* and *Adobe Pagemaker* still work with renditions, but they give authors a nicer interface to manipulate them. The user interface to the rendition (the file with formatting codes in it) is designed to look like the presentation (the finished paper product). We call this *What You See Is What You Get* (*WYSIWYG*) publishing. Since a rendition merely describes a presentation, it makes sense for the user interface to reflect the end-product.

1.1.1 *Formatting markup*

The form of typesetting notation that predates WYSIWYG (and is still in use today) is called *formatting markup*. Consider an analogy: you might submit a manuscript to a human typesetter for publication. Imagine it had no formatting, not even paragraphs or different fonts, but rather was a single continuous paragraph that was "marked up" with written instructions for how it should be formatted. You could write very precise instructions for layout: "Move this word over two inches. Bold it. Move the next work beside it. Move the next word underneath it. Bold it. Start a new line here." and so forth.



Fourscore and seven
years ago our fathers
brought forth on this
continent a new nation,
conceived in liberty,
and dedicated to the
propositions that all
men are created equal.
Now we are engaged in a
great civil war,
testing whether that
nation, or any nation

Figure 1-1 A manuscript "marked up" by hand

Formatting markup is very much the same. We "circle" text with instructions called *tags* or *codes* (depending on the particular formatting markup language). Here is an example of markup in one popular formatting markup language called LaTeX.

Example 1-1. A document with formatting markup

This is a marked up document. It contains words that are {\it italicized}, {\bf bold faced}, {\small small} and {\large large}.

In this markup language, the curly braces describe the extent of the formatting. So the italics started with the "\it" command extend until the end of the word "italicized". Because the markup uses only ordinary characters on typical keyboards, it can be created using existing text editors instead of special word processors (those came later).

1.1.2 *Generalized markup*

This process is adequate if your only goal is to type documents into the computer, describe a rendition and then print them. Around the late sixties, people started wanting to do more with their documents. In particular, IBM asked Charles Goldfarb (the name may sound familiar) to build a system for storing, finding, managing, and publishing legal documents.

Goldfarb found that there were many systems within IBM that could not communicate with each other. Each of them used a different command language. They could not read each other's files, just as you may have had trouble loading WordPerfect files into Word. The problem then, as now, was that they all had a different *representation* (sometimes also called a *file format*) for the information.

1.1.2.1 Common document representation

In the late sixties, Goldfarb and two other IBM researchers, Ed Mosher and Ray Lorie, set out to solve this problem. The team recognized three important facts. First, the programs needed to support a common document representation.

That part is easy to understand. Tools cannot work together if they do not speak the same language. As an analogy, consider the popularity of Latin terms in describing chemical and legal concepts and categories. To a certain extent, chemists and lawyers have chosen Latin as a common language for their fields. It made sense in the text processing context that the common language should be some form of markup language, because markup was well understood and very compatible with existing text editors and operating systems.

1.1.2.2 Customized document types

Second, the three realized that the common format should be *specific* to legal documents.

This is a little more subtle to grasp, but vital to understanding XML. The team could have invented a simple language, perhaps similar to the representation of a standard word processor, but that representation would not have allowed the sophisticated processing that was required. Lawyers and scientists both use Latin, but they do not use the same terminology. Rather they use Latin words as building blocks to create domain specific vocabularies (e.g. "habeas corpus", "ferruginous"). These domain specific vocabularies are even more important when we are describing documents to computers.

Computers are dumb

Usually we take for granted that computers are not very good at working with text and documents. We would never, for instance, ask a computer to search our hard disk and return a document that was a "letter" document, that was to "Martha" and that was about "John Smith's will". Even though this example seems much simpler than something a lawyer or chemist would run into, the fundamental problems are the same.

Most people recognize that the computer is completely incapable of understanding the concepts of "letter", "Martha" or "a will". Instead we might tell it to search for those words, and hope that we had included them all in the document. But what would happen if the system that we wanted to search was massive? It might turn up hundreds of unrelated documents. It might return documents that contained strings like "Martha, will you please write me a letter and tell me how John is doing?"

The fundamental problem is that the computer does not in any way understand the text. The solution is to teach the computer as much about the document as possible. Of course the computer will not understand the text in any real sense, but it can pretend to, in the same way that it pretends to understand simple data or decimal numbers[1]. We can make this possible by reducing the complexity of the document to a few structural *elements* chosen from a common vocabulary.

But computers can be trained

Once we "teach" computers about documents, we can also program them to do things they would not have been able to otherwise. Using their new

[1] We hope we haven't disillusioned anyone here. Computers may seem to know everything about math, but it is all a ruse. As far as they are concerned, they are only manipulating zeros and ones.

"understanding" they can help us to navigate through large documents, organize them, and automatically format the documents for publication in many different media, such as hypertext, print or tape.

In other words, we can get them to process text for us! The range of things we can get them to do with the documents is much wider than what we would get with WYSIWYG word processors or formatting markup.

Let us go back to the analogy of the typesetter working with a document marked up with a pen on paper to see why this is so powerful.

Imagine if we called her back the next day and told her to "change the formatting of the second chapter". She would have a lot of trouble mentally translating the codes for presentation back into high level constructs like sections and paragraphs.

To her, a title would only look like a line of text with a circle around it and instructions to make it italicized and 18 point. Making changes would be painful because recognizing the different logical constructs would be difficult. She probably could eventually accomplish the task by applying her human intuition and by reading the actual text. But computers do not have intuition, and cannot understand the text. That means that they cannot reliably recognize logical structure based totally on formatting. For instance they cannot reliably distinguish an italicized, 18 point title from an italicized, 18 point warning paragraph.

Even if human beings were consistent in formatting different types of documents (which we are not) computers would still have trouble. Even in a single document, the same formatting can mean two different things: italics could represent any kind of emphasis, foreign words, certain kinds of citations or other conventions.

Abstractions and renditions

Computers are not as smart as we are. If we want the computer to consider a piece of text to be written in a foreign language (for instance for spell checking purposes) then we must label it explicitly `foreign-language` and not just put it in italics! We call "foreign language" the *abstraction* that we are trying to represent, and we call the italics a particular rendition of the abstraction.

Formatting information has other problems. It is specific to a particular use of the information. Search engines cannot do very interesting searching on italics because they do not know what they mean. In contrast, the search engine could do something very interesting with `citation` elements: it could return a list of what documents are cited by other documents.

Italics are a form of markup specific to a particular application: formatting or printing. In contrast, the citation element is markup that can be used by a variety of applications. That is why we call this form of structural markup *generalized markup*. Generalized markup is the alternative to either formatting markup or WYSIWYG (lampooned by XML users as What You See is *All* You Get). Generalized markup is about getting more.

Because of the ambiguity of formatting, XML users typically do not bother to encode the document's presentational features at all, though XML would allow it. We are not interested, for instance, in fonts, page breaks and bullets. This formatting information would merely clutter up our abstract document's representation. Although typographic conventions allow the computer to print out or display the document properly, we want our markup to do more than that.

Stylesheets

Of course we must still be able to generate high quality print and online renditions of the document. Your readers do not want to read XML text directly. Instead of directly inserting the formatting commands in the XML document, we usually tell the computer how to generate formatted renditions *from* the XML abstraction.

For example in a print presentation, we can make the content of `TITLE` elements bold and large, insert page breaks before the beginning of chapters, and turn emphasis, citations and foreign words into italics. These rules are specified in a file called a *stylesheet*. The stylesheet is where human designers can express their creativity and understanding of formatting conventions. The stylesheet allows the computer to automatically convert the document from the abstraction to a formatted rendition.

We could use two different stylesheets to generate online and print renditions of the document. In the online rendition, there would be no page breaks, but cross-references would be represented as clickable hypertext links. Generalized markup allows us to easily produce high-quality print and online renditions of the same document.

This may well turn out to be the feature of XML that will save organizations and individuals the most money in the near future. We can even use two different stylesheets in the same medium. For instance, the computer could format the same document into several different styles (e.g. "New

York Times" style vs. "Wired Magazine") depending on the expressed preferences of a Web surfer, or even based on what Internet Service Provider they use.

We can also go beyond just print and online formatting and have our document be automatically rendered into braille or onto a text-to-speech machine. Generalized markup is highly endorsed by those who promote the *accessibility* of information to the visually impaired. XML should be similarly useful to those who want to widen the use of the Web.

Generalized markup documents are also "future-proof". They will not have to be redone to take advantage of future technologies. Instead, new stylesheets can be created to render existing documents in new ways.

Future renditions of documents might include three-dimensional virtual reality worlds where books are rendered as buildings, chapters as rooms and the text as wallpaper! Once again, the most important point is that these many different renditions will be possible without re-encoding the document. There are millions of SGML documents that predate the Web, but many of them are now published on it.

Typically, they were republished in HTML without changing a single character of the SGML source's markup or data, or editing a single character of the generated HTML. The same will be true of the relationship between XML and all future representations.

The key is abstraction. SGML and XML can represent abstractions, and from abstractions you can easily create any number of renditions. This is a fact well-known to the world's database programmers, who constantly generate new renditions – reports and forms – from the same abstract data.

Element types

Enough hype about generalized markup! You probably want to know what it looks like. To mark up a letter, we could identify the components of the letter like this:

Example 1-2. A simple memo

```
<to>Charles Goldfarb</to>
<from>Paul Prescod</from>
<re>John Smith's will</re>

<p>John Smith wants to update his will. Another wife left him.</p>
```

This text would be part of an XML document. The markup identifies components, called elements, of the document in ways that the computer can understand. The start-tag "<to>" marks the beginning of an element and the end-tag "</to>" marks the end of the element. Each element is an instance of an *element type*, such as "to", "from", "re" and "p".

If you use an XML-aware, you may never work with markup at the textual tag level, but you would still annotate sections of the document in this way (using whatever graphical interface the word processor provides).

Instead of each element type describing a formatting construct, each one instead describes the logical role of its elements – the *abstraction* it represents. The goal is for the abstraction to be descriptive enough and suitably chosen so that particular uses of the document (such as printing, searching and so forth) can be completely automated as computer processes acting on the elements.

For instance, we can search for a document that is "to" Martha, about ("re") John Smith's will. Of course the computer still does not understand the human interaction and concepts of sender and receiver, but it does know enough about the document to be able to tell me that in a "to" element of this particular document, the word "Martha" appears. If we expanded the letter a little to include addresses and so forth, we could also use an appropriate stylesheet to print it as a standard business letter.

Documents and databases

We can make our letter example even more precise and specific:

Example 1-3. Another letter

```
<to>Martha</to>
<from>Paul</to>
<re><customer-name>John Smith</customer-name>
    <customer-number>802-31348-5749</customer-number>
    <document-type-request>will</document-type-request>
</re>

<p>John Smith wants to update his will. Another wife left him.</p>
```

If you are familiar with databases, you might recognize that this looks database-ish in the sense that the customer number could be stored in a special index and you could easily search and sort this document based on customer numbers, document type requests and so forth.

But you can only do this sort of thing if your letter processing system understands your company's concepts of customer-numbers and your documents consistently provide the information. In other words, you must define your own set of element types just as the IBM team did.

In fact, many people have noticed that XML documents resemble traditional relational and object database data in many ways. Once you have a language for rigorously representing documents, those documents can be treated more like other forms of data.

But the converse is also true. As we have described, structured documents have many features in common with databases. They can preserve the abstract data and prevent it from being mingled with rendition information.

Furthermore, you can actually use this structured markup to represent data that is not what we would traditionally think of as documents, but too complex to be handled in conventional databases. In this brave new world, DNA patterns are data, and so are molecular diagrams and virtual reality worlds. In other words, generalized markup allows us to blow the doors off the word "document" and integrate diverse types of data. This database-ization of documents and document-ization of data is one of the major drivers of the XML excitement. Prior to XML, the Web had no standard data interchange format for even moderately complex data.

It may not have been obvious to its early, publishing-oriented adapters, that SGML would change the entire world of databases and electronic data interchange (EDI). But SGML's unique usefulness as a data interchange representation was a direct consequence of this second decision – to make SGML extensible through a customized vocabulary (set of element types).

1.1.2.3 Rule-based markup

The IBM team's third realization was that if computer systems were to work with these documents reliably, the documents would have to follow certain rules.

For instance a courtroom transcript might be required to have the name of the judge, defendant, both attorneys and (optionally) the names of members of the jury (if there is one). Since humans are prone to make mistakes, the computer would have to enforce the rules for us.

In other words the legal markup language should be specified in some formal way that would restrict elements appropriately. If the court stenographer tried to submit a transcript to the system without these elements being properly filled in, the system would check its *validity* and complain that it was *invalid*.

Of course, court transcripts have a different structure from wills, which in turn have a different structure from memos. So you would need to rigorously define what it means for each type of document to be valid. In SGML terminology, each of these is a *document type* and the formal definition that describes each type is called a *document type definition* (DTD).

Once again we can see why it is so important that the language provide us with the flexibility to choose our own vocabulary (our own set of element types). After all, the constraints that we apply must be described in terms of those element types. We use the word *document type* to refer both to a vocabulary and the constraints on its use.

Once again, this concept is very common in the database world. Database people typically have several layers of checking to guarantee that improper data cannot appear in their databases. For instance syntactic checks guarantee that phone numbers are composed of digits and that people's names are not. Semantic checks ensure that business rules are followed (such as "purchase order numbers must be unique"). The database world calls the set of constraints on the database structure a *schema*. In their terminology, DTDs are schemas for documents.

Once you have a document type worked out, you can describe for the computer how to print or display documents that conform to it with a *stylesheet*. So you might say that the address line in memos would be bolded, or that there should be two lines between speeches in a court transcript. These processes can work reliably because documents are constrained by the document type definition.

For instance, a letter cannot have a postscript ("P.S.") at the beginning of the document nor an address at the end. Because there is no convention for formatting such a letter, a stylesheet would not typically do a good job with it. In fact, it might crash, as some word processors do when they try to load corrupted documents. The document type definition protects us from this.[1]

In 1969, the IBM team developed a language that could implement their vision of markup that was not specific to a particular system. They called it the *Generalized Markup Language* (which, not coincidentally, has the same initials as the names Goldfarb, Mosher and Lorie).

[1] Of course, computer programmers will always invent new excuses for crashing software.

However, it wasn't until 1974 that Goldfarb proved the concept of a "validating parser", one that could read a document type definition and check the accuracy of markup, without going to the expense of actually processing a document. As he recalls it: "At that point SGML was born – although it still had a lot of growing up to do."

Between 1978 and 1986, Goldfarb acted as technical leader of a team of users, programmers and academics that developed his nascent invention into the robust International Standard (ISO 8879) they called the *Standard Generalized Markup Language.*

That team, with many of the same players still involved, is now JTC1/WG4, which continues to develop SGML and related standards. Two of the most important are *HyTime*, which standardizes the representation of hyperlinking features, and DSSSL, which standardizes the creation of stylesheets.[1]

The SGML standard took a long time to develop, but arguably it was still ahead of the market when it was created. Over those years, the basic concepts of GML were broadened to support a very wide range of applications. Although GML was always extensible and generalized, the SGML standard added many features and options, many intended for niche markets. But the niches had to be catered for: some of the niche users have document collections that rival the Web in size!

By the time it was standardized in 1986, SGML had become large, intricate and powerful. In addition to being an official International Standard, SGML is the defacto standard for the interchange of large, complex documents and has been used in domains as diverse as programming language design and airplane maintenance.

1.2 | HTML and the Web

In 1989, a researcher named Tim Berners-Lee proposed that information could be shared within the CERN European Nuclear Research Facility using hyperlinked text documents. He was advised to use an SGML-ish

[1] Knowing the full names probably won't help much, but just in case, HyTime is short for "Hypermedia/Time-based Structuring Language" and DSSSL (pronounced "dis-sal") is short for "Document Style Semantics and Specification Language". We warned you that it wouldn't help much.

syntax by a colleague named Anders Berglund, an early adopter of the new SGML standard. They started from a simple example document type in the SGML standard[1] and developed a hypertext version called the *Hypertext Markup Language* (HTML).

Relative to the 20 year evolution of SGML, HTML was developed in a hurry, but it did the job. Tim called his hypertext system the *World Wide Web* and today it is the most diverse, popular hypertext information system in existence. Its simplicity is widely believed to be an important part of its success. The simplicity of HTML and the other Web specifications allowed programmers around the world to quickly build systems and tools to work with the Web.

HTML inherited some important strengths from SGML. With a few exceptions, its element types were generalized and descriptive, not formatting constructs as in languages like TeX and Microsoft Word. This meant that HTML documents could be displayed on text screens, under graphical user interfaces, and even projected through speakers for the sight impaired.

HTML documents used SGML's simple angle bracket convention for markup. That meant that authors could create HTML documents in almost any text editor or word processor. The documents are also compatible with almost every computer system in existence.

On the other hand, HTML only uses a fixed set of element types. As we discussed before, no one document type can serve all purposes, so HTML only adopted the first of GML's revelations, that document representations must be standardized. It is not extensible and therefore cannot be tailored for particular document types, and it was not very rigorously defined until years after its invention. By the time HTML was given a formal DTD, there were already thousands of Web pages with erroneous HTML.[2]

[1] That DTD was based on the very first published DTD, from a 1978 IBM manual written by Goldfarb, derived in turn from work that he and Mosher had done in the early 70's.

[2] Today there are tens of thousands with misleading or downright erroneous informational content, so perhaps bad HTML is not that big a problem in practice.

1.2.1 *HTML gets extended – unofficially!*

As the Web grew in popularity many people started to chafe under HTML's fixed document type. Browser vendors saw an opportunity to gain market share by making incompatible extensions to HTML. Most of the extensions were formatting commands and thus damaged the Web's interoperability. The first golden rule, standardization was in serious danger.

For instance Netscape's popular `CENTER` element cannot be "pronounced" in a text to speech converter. A `BLINK` element cannot be rendered on some computers. Still, this was a fairly understandable reaction to HTML's limitations.

One argument for implementing formatting constructs instead of abstractions is that there are a fixed number of formatting constructs in wide use, but an ever growing number of abstractions. Let's say that next year biologists invent a new formatting notation for discussing a particular type of DNA. They might use italics to represent one kind of DNA construct and bold to represent another. In other words, as new abstractions are invented, we usually use existing formatting features to represent them. We have been doing this for thousands of years, and prior to computerization, it was essentially the only way.

We human readers can read a textual description of the meanings of the features ("in this book, we will use Roman text to represent...") and we can differentiate them from others using our reasoning and understanding of the text. But this system leaves computers more or less out of the loop.

For instance superscripts can be used for trademarks, footnotes and various mathematical constructs. Italics can be used for references to book titles, for emphasis and to represent foreign languages. Without generalized markup to differentiate, computers cannot do anything useful with that information. It would be impossible for them to translate foreign languages, convert emphasis to a louder voice for text to speech conversion, or do calculations on the mathematical formulae.

1.2.2 *The World Wide Web reacts*

As the interoperability and scalability of the Web became more and more endangered by proprietary formatting markup, the World Wide Web Consortium (headed by the same Tim Berners-Lee) decided to act. They attacked the problem in three ways. First, they decided to adopt the SGML convention for attaching formatting to documents, the stylesheet.

They invented a simple HTML-specific stylesheet language called *Cascading Style Sheets* (CSS) that allowed people to attach formatting to HTML documents without filling the HTML itself with proprietary, rendition-oriented markup.

Second, they invented a simple mechanism for adding abstractions to HTML. We will not look at that mechanism here, because XML makes it obsolete. It allowed new abstractions to be invented but provided no mechanism for constraining their occurrence. In other words it addressed two of GML's revelations: it brought HTML back to being a single standard, more or less equally supported by the major vendors, and it allowed people to define arbitrary extensions (with many limitations).

But they knew that their stool would not stand long on two of its three legs. The (weakly) extensible HTML and CSS are only stopgaps. For the Web to move to a new level, it had to incorporate the third of SGML's important ideas, that document types should be formally defined so that documents can be checked for validity against them.

Therefore, the World Wide Web Consortium decided to develop a subset of SGML that would retain SGML's major virtues but also embrace the Web ethic of minimalist simplicity. They decided to give the new language the catchy name Extensible Markup Language (XML). They also decided to make related standards for advanced hyperlinking and stylesheets.

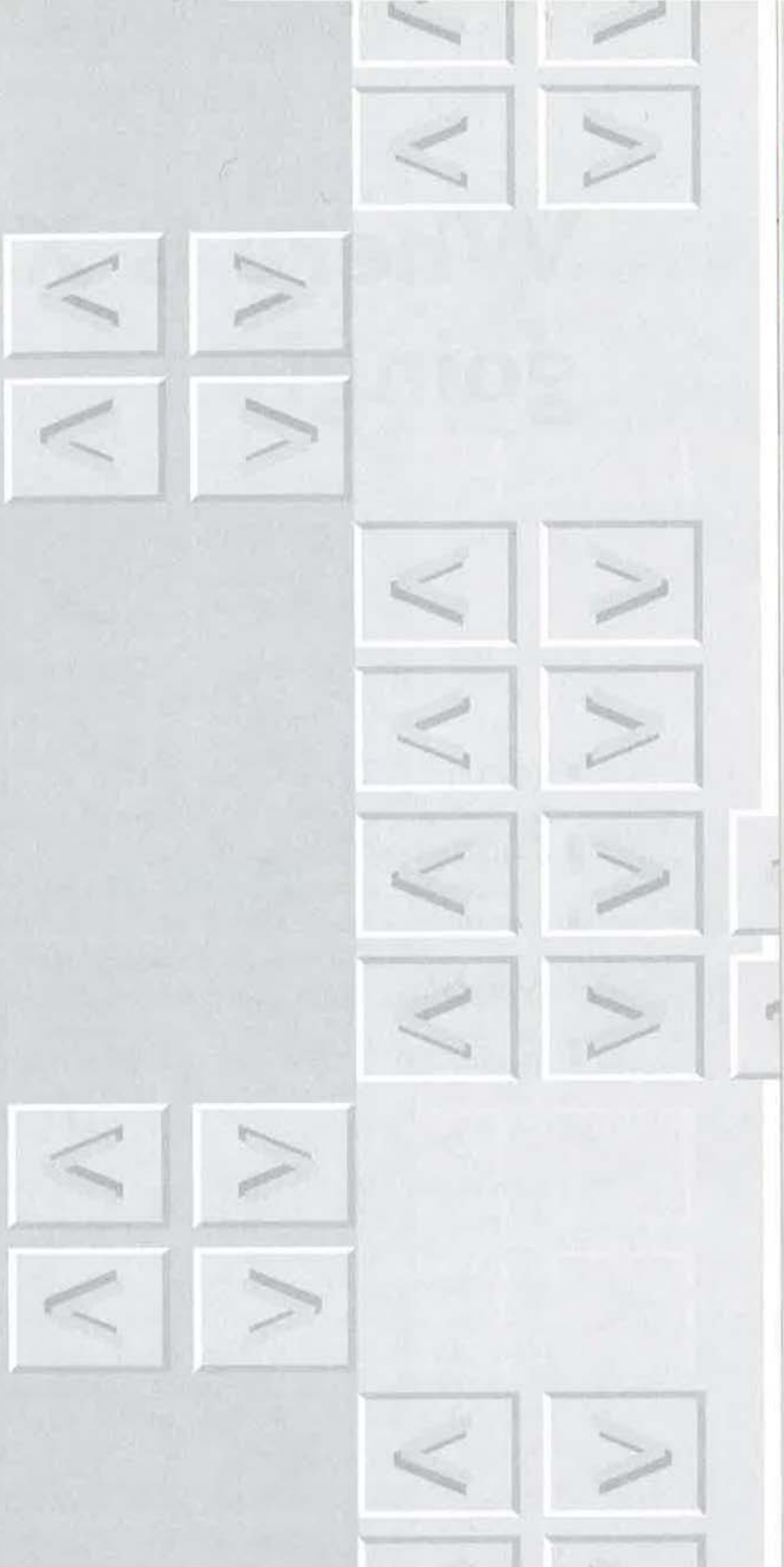
The first, called the Extensible Linking Language (XLink), is inspired by HyTime, the ISO standard for linking SGML documents, and by the Text Encoding Initiative, the academic community's guidelines for applying SGML to scholarly applications.

The second, called the Extensible Style Language is a combination of ideas from the Web's Cascading Style Sheets and ISO's DSSSL standard.[1]

[1] This description necessarily presented as linear, straightforward, and obvious a process that was actually messy and at times confusing. It is fair to say that there were many people outside the World Wide Web Consortium who had a better grasp on the need for XML than many within it, and that various member corporations "caught on" to the importance of XML at different rates.

1.3 | Conclusion

Now we've seen the origins of XML, and some of its key ideas. Unlike lots of other "next great things" of the high-tech world, XML has solid roots and a proven track record. You can have confidence in XML because the particular subset of SGML that is XML has been in use for a dozen years.



Chapter 2

The XML effort is new ground in many senses. The Web has never before had access to the new features that XML offers. It will take a while for the Web culture to understand the strengths (and weaknesses) of the new language and learn how to properly deploy it. Still, XML is already becoming a building block for the next generation of Web applications and specifications.

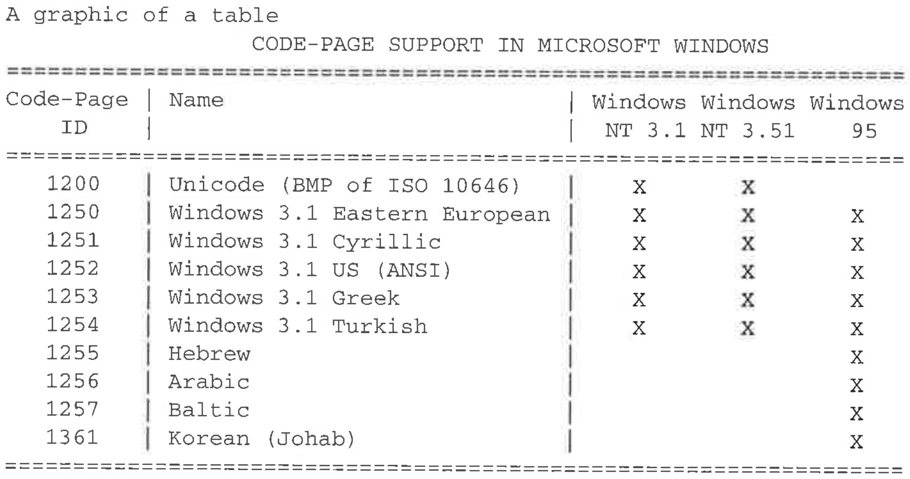
2.1 | Beyond HTML

XML was originally conceived as a big brother to HTML. As its name implies, XML can be used to extend HTML or even define whole new languages completely unlike HTML. At first, the thousands of authors accustomed to HTML will probably use it as a mere HTML extension and slowly grow into its more powerful abilities.

For instance, a company might want to offer technical manuals on the Web. Many manuals have a formatting for tables (e.g., a table listing a software product's supported languages) and repeat the formatting on several tables in the manual (perhaps once per program in a package). The formatting of these tables can be very intricate.

For instance the rows may be broken into categories with borders between them. The title of each column and row might be in a particular font and color. The width of the columns might be very precisely described. The final row ("the bottom line") might be colored. HTML could provide the formatting markup that the layout would require, but it would require a lot of duplication. In fact, it would be such a hassle that most companies would choose to use a graphic or an *Adobe Portable Document Format* (PDF) file instead.

To demonstrate how XML can help, we will use an example table from the specification for HTML tables. We will simplify the example somewhat, but the XML solution will still be shorter (in characters) and easier to read.



A graphic of a table

CODE-PAGE SUPPORT IN MICROSOFT WINDOWS

Code-Page ID	Name	Windows NT 3.1	Windows NT 3.51	Windows 95
1200	Unicode (BMP of ISO 10646)	X	X	
1250	Windows 3.1 Eastern European	X	X	X
1251	Windows 3.1 Cyrillic	X	X	X
1252	Windows 3.1 US (ANSI)	X	X	X
1253	Windows 3.1 Greek	X	X	X
1254	Windows 3.1 Turkish	X	X	X
1255	Hebrew			X
1256	Arabic			X
1257	Baltic			X
1361	Korean (Johab)			X

Figure 2-1 A Formatted Table (based on an example in the HTML 4.0 Recommendation)

If there are many of these tables the cumulative effort of doing this manual work can add up to a large burden, especially since it must be maintained as products change. Even with an HTML authoring tool, you will probably have to do the layout manually, over and over again. As if this internal expense was not disturbing enough, every person who reads the annual report over the Web must download the same formatting informa-

Example 2-1. The HTML Markup to Implement the Table (based on an example in the HTML 4.0 Recommendation)

```
<TABLE border="2" frame="hsides" rules="groups">
<CAPTION>CODE-PAGE SUPPORT IN MICROSOFT WINDOWS</CAPTION>
<COLGROUP align="center">
<COLGROUP align="left">
<COLGROUP align="center" span="2">
<COLGROUP align="center" span="3">
<THEAD valign="top"><TR><TH>Code-Page<br>ID<TH>Name
<TH>Windows<br>NT 3.1<TH>Windows<br>NT 3.51<TH>Windows<br>95
<TBODY>
<TR><TD>1200<TD>Unicode (BMP of ISO/IEC-10646)<TD>X<TD>X<TD>
<TR><TD>1250<TD>Windows 3.1 Eastern European<TD>X<TD>X<TD>X
<TR><TD>1251<TD>Windows 3.1 Cyrillic<TD>X<TD>X<TD>X
<TR><TD>1252<TD>Windows 3.1 US (ANSI)<TD>X<TD>X<TD>X
<TR><TD>1253<TD>Windows 3.1 Greek<TD>X<TD>X<TD>X
<TR><TD>1254<TD>Windows 3.1 Turkish<TD>X<TD>X<TD>X
<TR><TD>1255<TD>Hebrew<TD><TD><TD>X
<TR><TD>1256<TD>Arabic<TD><TD><TD>X
<TR><TD>1257<TD>Baltic<TD><TD><TD>X
<TR><TD>1361<TD>Korean (Johab)<TD><TD><TD>X</TABLE>
```

tion row after row, column after column, table after table, year after year. Right thinking Web page authors will understand that this situation is not good. The repetition leads to longer download times, congested servers, dissatisfied customers and perhaps irate managers.

The XML solution would be to invent a simple extension to HTML that is customized to the needs of the manual. It would have table elements that would only require data that varies from table to table. None of the redundant formatting information would be included. We would then use a sophisticated stylesheet to add that information back in. The beauty of the stylesheet solution is that the formatting information is expressed only in one place. Surfers only have to download that once. Also, if your company decides to change the style of the tables, all of them can be changed at once merely by changing the stylesheet. Here is what that might look like:

The difference between this XML version and the HTML version is not as dramatic as in some examples, but the XML version is clearer, has fewer lines and characters and is easier to maintain. More important, the stylesheet can choose to format this in many different ways as time goes by and tastes change. All the XML version represents is the actual information about Windows code pages, not the tabular format of a particular presentation of it.

Example 2-2. XML Version of the Table

```
<CODE-PAGE-TABLE>
<CP NUM="1200" NAME="Unicode (BMP of ISO/IEC-10646)"
   PLATFORMS="NT3.1 NT3.51"/>
<CP NUM="1250" NAME="Windows 3.1 Eastern European"
   PLATFORMS="NT3.1 NT3.51 WIN95"/>
<CP NUM="1251" NAME="Windows 3.1 Cyrillic"
   PLATFORMS="NT3.1 NT3.51 WIN95"/>
<CP NUM="1252" NAME="Windows 3.1 US (ANSI)"
   PLATFORMS="NT3.1 NT3.51 WIN95"/>
<CP NUM="1253" NAME="Windows 3.1 Greek"
   PLATFORMS="NT3.1 NT3.51 WIN95"/>
<CP NUM="1254" NAME="Windows 3.1 Turkish"
   PLATFORMS="NT3.1 NT3.51 WIN95"/>
<CP NUM="1255" NAME="Hebrew"
   PLATFORMS="WIN95"/>
<CP NUM="1256" NAME="Arabic"
   PLATFORMS="WIN95"/>
<CP NUM="1257" NAME="Baltic"
   PLATFORMS="WIN95"/>
<CP NUM="1261" NAME="Korean (Johab)"
   PLATFORMS="NT3.1 NT3.51 WIN95"/>
```

One thing to note is that the extra download of a stylesheet does take time. It makes the most sense to move formatting into a stylesheet when that formatting will be used on many pages or in many parts of the same page. The goal is to amortize the cost of the download over a body of text. A similar caveat applies to the time it takes to make the stylesheet and design the table elements. Doing so for a single table would probably not be cost effective. Our example above basically shifts the complexity from the document to the stylesheet, on the presumption that there will probably be many documents (or at least many tables) for every stylesheet. In general, XML is about short term investment in long term productivity.

Once you have made that investment you can sometimes realize more radical productivity gains than you first intended. Imagine that you use XML tables to publish the financial information in your company's annual report. Your accountants may be able to use their software's report writing feature to directly transfer accounting information into the XML table. This can save one more opportunity for typos between the accountants' printout and the Web author's keyboard. There might also be opportunities for automation at the other end of the spectrum. Other software might transform the XML table directly into a format required for submission to some government agency.

2.2 | Database publishing

The last example hints at the way XML can interact with systems that are not typically associated with documentation. As documents become more structured they can become integrated with the other structured data in an organization. Some of the same techniques can be used to create them (such as report writing software or custom graphical user interfaces) and some of the same software will be able to read them (such as spreadsheets and database software). One particularly popular application of XML will surely be the publishing of databases to the Web.

Consider for instance a product database, used by the internal ordering system of a toy manufacturer. The manufacturer might want the database to be available on the Web so that potential clients would know what was available and at what price. Rather than having someone in the Web design department mark up the data again, they could build a connection between their Web server and their database using the features typically built into Web servers that allow those sorts of data pipes. The designers could then make the products list beautiful using a stylesheet. Pictures of the toys could be supplied by the database. In essence, the Web site would be merely a view on the data in the database. As toys get added and removed from the database, they will appear and disappear from the view on the Website. This mechanism also gives the Website maintainer the freedom to update the "look and feel" of the Website without dealing with the database or the plumbing that connects it to the Web server!

XML is also expected to become an important tool for interchange of database information. Databases have typically interchanged information using simple file formats like one-record per line with semi-colons between the fields. This is not sufficient for the new object-oriented information being produced by databases. Objects must have internal structure and links between them. XML can represent this using elements and attributes to provide a common format for transferring database records between databases. You can imagine that one database might produce an XML document representing all of the toys the manufacturer produces and that document could be directly loaded into another database either within the company or at a customer's site. This is a very interesting way of thinking about documents, because in many cases human beings will never see them. They are documents produced by and for computer software.

Example 2-3. A products database in XML

```
<TOYS>
<ITEM>
<TITLE>GI John</TITLE>
<MANUFACTURER>War Toys Inc.</MANUFACTURER>
<PRICE>50.95</PRICE>
<IN-STOCK>3000</IN-STOCK>
</ITEM>
<ITEM>
<TITLE>Leggo!</TITLE>
<MANUFACTURER>Grips R US</MANUFACTURER>
<PRICE>64.95</PRICE>
<IN-STOCK>2000</IN-STOCK>
</ITEM>
<ITEM>
<TITLE>Hell On Wheels</TITLE>
<MANUFACTURER>Li'l Road Warriors</MANUFACTURER>
<PRICE>150.95</PRICE>
<IN-STOCK>3200</IN-STOCK>
</ITEM>
</TOYS>
```

2.3 | Electronic commerce

Presume that a retailer decides that it wants to start selling a line of toys from the database. They might contact the manufacturer to organize the sale. The two could agree on an XML-based product-request message format and formalize it in an XML document type. In fact, there might already be an industry standard XML document type appropriate for the task. Once that has been chosen, orders for the part can be sent automatically from the purchaser's computer to the supplier's. This sort of electronic commerce has been possible for years, but XML allows it to be easily standardized, highly extensible and wired into the backbone technologies of the Internet. The easy availability of the software and standards will allow much smaller organizations to use electronic commerce.

2.4 | Metadata

There is a special type of data that interests the larger Web publishers. It is called metadata: information about information. XML is the basis for

Example 2-4. An order for a Toy

```
<Toy-Order>
<Order-No>967634</Order-No>
<Message-Date>19961002</Message-Date>
<Buyer-EAN>5412345000176</Buyer-EAN>
<Toy><Number>523953-432</Number><Quantity>18</Quantity></Toy>
<Toy><Number>438312-716</Number><Quantity>13</Quantity></Toy>
<Toy><Number>232332-136</Number><Quantity>23</Quantity></Toy>
</Toy-Order>
```

metadata standards such as Microsoft's Channel Definition Format (CDF) for describing "Web Push Channels", Netscape's Meta Content Framework (MCF) for tracking information about Web sites and the *Platform for Internet Content Selection* which allows the filtering of inappropriate material from computer screens based on external descriptions of content. These applications are called metadata because they are used to describe other information resources. The "violent content" label on a video tape is a perfect example of metadata. The data provided, "violent content" describes the contents of the tape – it is data about data.

CDF describes things like about Web channels, such as their schedules and logos and can carry a description of the channel. This may sound familiar to you. If you think about it, you will notice that even TV Guide is metadata! Some future online version might use XML. Netscape's MCF can describe things like who is in charge of a Web page, what other pages are related to it, how they are related and so forth.

Example 2-5. Channel Description Format

```
<?XML version="1.0"?>
<CHANNEL HREF="http://www.rocktv.com/channels">
<ABSTRACT>
RockTV is your 24-hour rock station! Nothing but geology,
geography and rock collecting. All day! All night!
</ABSTRACT>
</CHANNEL>
```

XML is convenient for these tasks for several reasons. It can be edited in standard text editors and specialized XML word processors. XML's syntax will be familiar to the millions of Web maintainers who must eventually learn to apply metadata. XML expresses the hierarchy and links of these documents nicely. It is also well suited to encoding the textual portions of specifications. For instance every channel will have a textual description hoping to convince you to subscribe. XML can allow these descriptions to use its hypermedia features to create very compelling displays.

The next step in the evolution of metadata on the Web is a standardized layer on top of XML called *Resource Description Framework* (RDF). RDF is still under development, but when it is finished, it will be an XML document type for metadata that will be extensible at the metadata level as XML is at the document level. What that means is that RDF documents will be able to describe new relationships between documents, images and other Web resources. This will allow new relational vocabularies to be developed just as XML document types allow new markup vocabularies. Older metadata standards like PICS and CDF will eventually be revamped in terms of XML and RDF. In one sense, this sounds very complicated: PICS is based on RDF which is based on XML. But on the other hand, it will not be so complicated in practice. PICS and CDF will have a set of element types that you must learn to apply according to the XML syntax described in this book. RDF, the middle layer, will only be visible to the wizards who invent new ways of cataloging, describing and organizing information – the librarians of the future.[1]

Example 2-6. Describing the owner of a document in RDF

```
<RDF:assertions href="http://www.bar.com/some.doc>
    <bib:author>
 <RDF:resource>
   <bib:name>John Smith</bib:name>
 <bib:email>john@smith.com</bib:email>
 <bib:phone>+1 (555) 123-4567</bib:phone>
      </RDF:resource>
 </bib:author>
</RDF:assertions>
```

2.5 | Science on the Web

Although the Web was originally invented in a physics laboratory for communication among physicists, it never developed into a great system for

[1] Luckily, the librarians of the present are very much involved in these standardization efforts.

communicating mathematical formulae. Markup for mathematics is more complex than it seems at first to non-mathematicians, and the mathematicians have not yet agreed exactly how they want to do it.

Suffice to say that there are some attempts to do math on the Web that attempt to do too much and others that do not do enough. The World Wide Web Consortium is working on a new XML-based language called *MathML*. Hopefully MathML will strike a good middle ground. MathML markup is demonstrated in Example 2-7 and a rendered formula is in Figure 2-2.

Example 2-7. MathML Markup for a Formula

```
<mrow>
  <mrow>
    <msup>
      <mi>x</mi>
      <mn>2</mn>
    </msup>
    <mo>+</mo>
    <mrow>
      <mn>4</mn>
      <mo>&invisibletimes;</mo>
      <mi>x</mi>
    </mrow>
    <mo>+</mo>
    <mn>4</mn>
  </mrow>
  <mo>=</mo>
  <mn>0</mn>
</mrow>
```

$$x^2 + 4x + 4 = 0$$

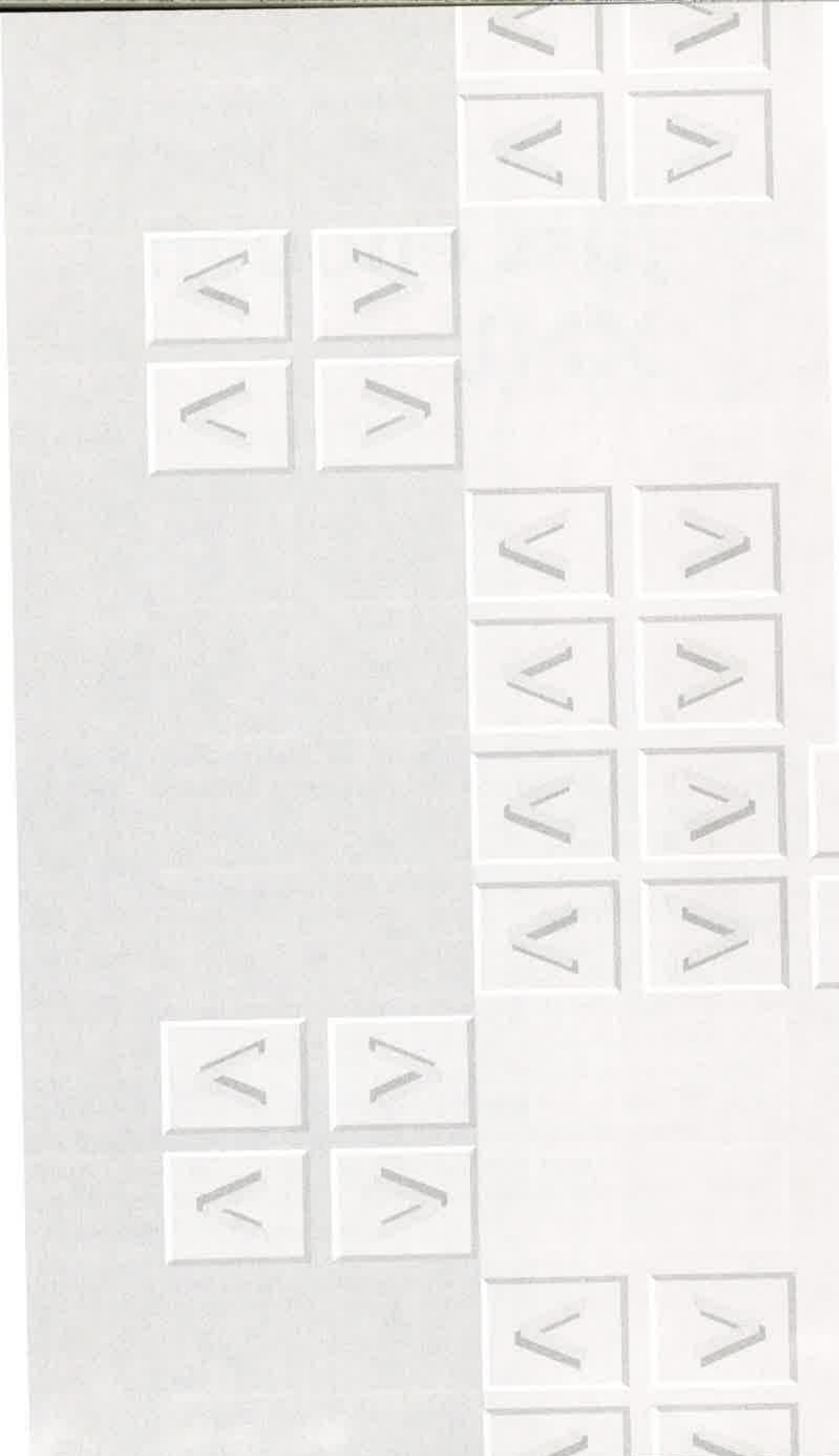
Figure 2-2 MathML Formula Rendered

The *Chemical Markup Language* (CML) is an XML-based language for describing the management of molecular information on computer networks. Using a Java viewer that is under development, users can view and manipulate molecules in 2 and 3 dimensions. *Bioinformatic Sequence*

Markup Language is a standard for encoding DNA, RNA and protein sequence information.

The specs for both of these applications are available on the CD-ROM that accompanies this book.

As you can see, XML is branching out into a wide variety of problem domains. Whatever your discipline, you should consider if there is some part of your workflow that could be made more efficient with standardization based on XML. In subsequent parts of this book, we will explore in detail a wide array of applications of the kinds we have been describing.



Just enough XML

- Elements
- Character set
- Entities
- Markup
- Document types

Chapter 3

In this chapter we will explore the fundamental concepts of XML documents and XML systems. If XML were a great work of literature then this chapter would be the Cliff notes. The chapter will introduce the ideas that define the language but will avoid the nitty gritty details (the *syntax*) behind the constructs. As a result, some concepts may remain slightly fuzzy because you will not be able to work with them "hands on". Later chapters will provide that opportunity.

This early presentation of these ideas will allow you to see XML's "big picture". We will do this by walking through the design process for an XML-like language. Hopefully by the end of the process, you will understand each of the design decisions and XML's overall architecture.

Our objective is to equip you with "just enough" XML to appreciate the application scenarios and tool descriptions in the following parts of the book, but being over-achievers we may go a little too far. Feel free to leave at any time to read about XML in the real world.

3.1 | The goal

First we should summarize what we are trying to achieve. In short, "What is XML used for?" XML is for the *digital representation* of documents. You probably have an intuitive feel for what a document is. We will work from your intuition.

Documents can be large and small (). Both a multi-volume encyclopedia and a memo can be viewed as documents. A particular volume of the encyclopedia can also be called a document. XML allows you to think of the encyclopedia whichever way will allow you to get your job done most efficiently. You'll notice that XML will give you these sorts of options in many places. XML also allows us to think of an email message as a document. XML can even represent the message from a police department's server to a police officer's handheld computer that reports that you have unpaid parking tickets.[1]

When we say that we want to *digitally represent* documents we mean that we want to put them in some kind of computer-readable code so that a computer can help us store, process, search, transmit, display and print them. In order for a computer to do useful things with a document, we are going to have to tell it about the structure of the document. This is our simple goal: to put the documents in a code that the computer can "understand" in-so-far as computers can understand anything.

XML documents can include pictures, movies and other multimedia, but we will not actually represent the multimedia components as XML. If you think of representation as a translation process, similar to language translation, then the multimedia components are the parts that we will leave in their "native language" because they have no simple translation into the "target language" (XML). We will just include them in their native formats as you might include a French or Latin phrase in an English text without explicit translation. Most pictures on the Web are files in formats called GIF or JPEG and most movies are in a format called MPEG. An XML document would just refer to those files in their native GIF, JPEG or MPEG formats. If you were transcribing an existing print document into XML, you would most likely represent the character-text parts as XML and the graphical parts in these other formats.

[1] Sorry about that.

3.2 | Elements: The logical structure

Before we can describe exactly how we are going to represent documents, we must have a model in our heads of how a document is structured. Most documents (for example books and magazines) can be broken down into components (chapters and articles). These can also be broken down into components (titles, paragraphs, figures and so forth). It turns out that just about every document can be viewed this way.

In XML, these components are called *elements*. Each element represents a logical component of a document. Elements can contain other elements and can also contain the words and sentences that you would usually think of as the text of the document. XML calls this text the document's *character data*. This hierarchical view of XML documents is demonstrated in Figure 3-1.

Markup professionals call this the *tree structure* of the document. The element that contains all of the others (e.g. `book`, `report` or `memo` is known as the *root element*. This name captures the fact that it is the only element that does not "hang" off of some other element.

The elements that are contained in the root are called its *sub-elements*. They may contain sub-elements themselves. If they do, we will call them *branches*. If they do not, we will call them *leaves*.

Thus, the `chapter` and `article` elements are branches (because they have sub-elements), but the `paragraph` and `title` elements are leaves (because they only contain character data).[1] The root element is also referred to as the *document element* because it holds the entire logical document within it. The terms *root element* and *document element* are interchangeable.

Elements can also have extra information attached to them called *attributes*. Attributes describe properties of elements. For instance a `CIA-record` element might have a security attribute that gives the security rating for that element. A CIA database might only release certain records to certain people depending on their security rating. It will not always be clear which aspects of a document should be represented with elements and which should be represented with attributes, but we will give some guidelines in Chapter 32, "Creating a document type definition", on page 448.

[1] You can see from this terminology that markup experts tend to have an environmentalist bent. The latest word sweeping the markup world is "grove", a term that recognizes that a single document may have multiple trees, for attributes (see below) as well as elements.

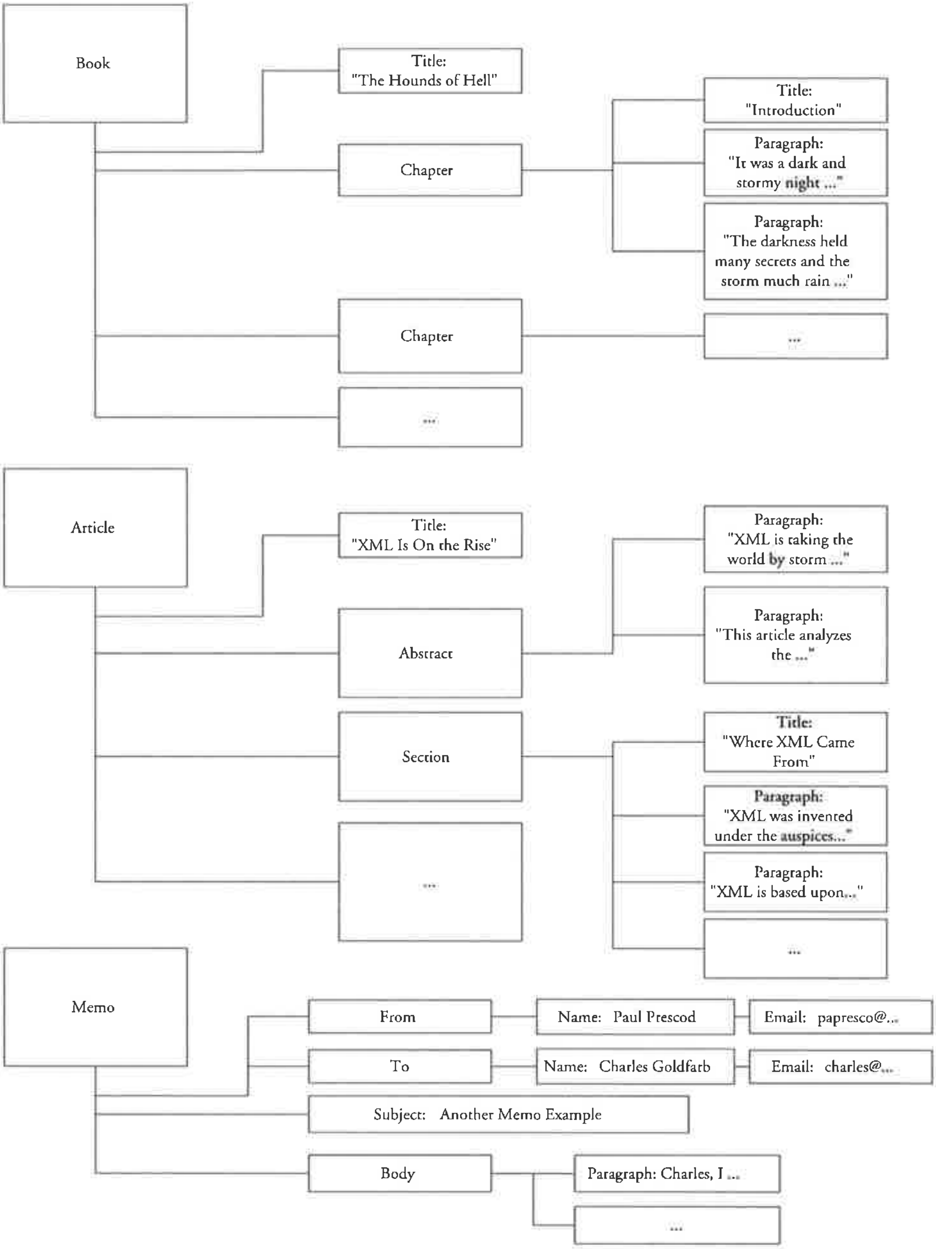

Book
Title: "The Hounds of Hell"
Chapter
Title: "Introduction"
Paragraph: "It was a dark and stormy night ..."
Paragraph: "The darkness held many secrets and the storm much rain ..."
Chapter
...
...
Article
Title: "XML Is On the Rise"
Abstract
Paragraph: "XML is taking the world by storm ..."
Paragraph: "This article analyzes the ..."
Section
Title: "Where XML Came From"
Paragraph: "XML was invented under the auspices..."
Paragraph: "XML is based upon..."
...
...
Memo
From
Name: Paul Prescod
Email: papresco@...
To
Name: Charles Goldfarb
Email: charles@...
Subject: Another Memo Example
Body
Paragraph: Charles, I ...
...

Figure 3-1 Hierarchical views of documents

Real-world documents do not always fit this *tree* model perfectly. They often have non-hierarchical features such as cross-references or hypertext links from one section of the tree to another. XML can represent these structures too. In fact, XML goes beyond the powerful links provided by HTML. More on this in 3.8, "Hyperlinking and Addressing", on page 45

3.3 | Unicode: The character set

Texts are made up of characters. If we are going to represent texts, then we must represent the characters that comprise them. So we must decide how we are going to represent characters at the bits and bytes level. This is called the *character encoding*. We must also decide what characters we are going to allow in our documents. This is the *character set*. A particularly restrictive character set might allow only upper-case characters. A very large character set might allow Eastern ideographs and Arabic characters.

If you are a native English speaker you may only need the fifty-two upper- and lower-case characters, some punctuation and a few accented characters. The pervasive *7 bit ASCII character set* caters to this market. It has just enough characters (128) for all of the letters, symbols, some accented characters and some other oddments. ASCII is both a character set *and* a character encoding. It defines what set of characters are available and how they are to be encoded in terms of bits and bytes.

XML's character set is Unicode, a sort of ASCII on steroids. Unicode includes thousands of useful characters from languages around the world.[1] However the first 128 characters of Unicode are compatible with ASCII and there is a character encoding of Unicode, *UTF-8* that is compatible with 7 bit ASCII. This means that at the bits and bytes level, the first 128 characters of UTF-8 Unicode and 7 bit ASCII are the same. This feature of *Unicode* allows authors to use standard plain-text editors to create XML immediately.

[1] It also includes some not-so-useful characters – there is an entire section dedicated to "dingbats" and there is a proposal to include "Klingon", the artificial language from *Star Trek*™.

3.4 | Entities: The physical structure

An XML document is defined as a series of characters. An XML processor starts at the beginning and works to the end. XML provides a mechanism for allowing text to be organized non-linearly and potentially in multiple pieces. The parser reorganizes it into the linear structure.

The "piece-of-text" construct is called an *entity*. An entity could be as small as a single character or as large as all the characters of a book.

Entities have *names*. Somewhere in your document, you insert an *entity reference* to make use of an entity. The processor replaces the entity reference with the entity itself, which is called the *replacement text*. It works somewhat like a wordprocessor macro.

For instance an entity named "sigma", might contain the name of a Greek character. You would use a reference to the entity whenever you wanted to insert the sigma character. An entity could also be called "introduction-chapter" and be a chapter in a book. You would refer to the entity at the point where you wanted the chapter to appear.

One of the ideas that excited Ted Nelson, the man who coined the word *hypertext*, was the idea that text could be reused in many different contexts automatically. An update in one place would propagate across all uses of the text. The feature of XML that allows this is called the external entity. External entities are often referred to merely as entities, but the meaning is usually clear from context. An XML document can be broken up into many files on a hard disk or objects in a database and each of them is called an entity in XML terminology. Entities could even be spread across the Internet. Whereas XML elements describe the document's logical structure, entities keep track of the location of the chunks of bytes that make up an XML document. We call this the physical structure of the document.

> **Note** *The unit of XML text that we will typically talk about is the entity.You may be accustomed to thinking about files, but entities do not have to be stored as files.*

For instance, entities could be stored in databases or generated on the fly by a computer program. Some file formats (e.g. a *zip* file) even allow multiple entities to reside in the same file at once. The term that covers all of these possibilities is entity, *not* file. Still, on most Web sites each entity will reside in a single file so in those cases external entities and files will functionally be the same. This setup is simple and efficient, but will not be sufficient for very large sites.

Entities' bread and butter occupation is less sexy than reusing bits of text across the Internet. But it is just as important: entities help to break up large files to make them editable, searchable, downloadable and otherwise usable on the ordinary computer systems that real people use. Entities allow authors to break their documents into workable chunks that can fit into memory for editing, can be downloaded across a slow modem and so forth.

Without entities, authors would have to break their documents unnaturally into smaller documents with only weak links between them (as is commonly done with HTML). This complicates document management and maintenance. If you have ever tried to print out one of these HTML documents broken into a hundred HTML files then you know the problem. Entities allow documents to be broken up into chunks without forgetting that they actually represent a single coherent document that can be printed, edited and searched as a unit when that makes sense.

Non-XML objects are referenced in much the same way and are called *unparsed entities.* We think of them as "data entities" because there is no XML markup in them that will be noticed. Data entities include graphics, movies, audio, raw text, PDF and anything else you can think of that is not XML (including HTML and other forms of SGML).[1] Each data entity has an associated *notation* that is simply a statement declaring whether the entity is a GIF, JPEG, MPEG, PDF and so forth.

Entities are described in all of their glorious (occasionally gory) detail in Chapter 33, "Entities: Breaking up is easy to do", on page 476

3.5 | Markup

We have discussed XML's conceptual model, the tree of elements, its strategy for encoding characters, Unicode, and its mechanism for managing the size and complexity of documents, entities. We have not yet discussed how

[1] Actually, a data entity could even contain XML, but it wouldn't be treated as part of the main XML document.

to represent the logical structure of the document and link together all of the physical entities.

Although there are XML word processors, one of the design goals of XML was that it should be possible to create XML documents in standard text editors. Some people are not comfortable with word processors and even those who are may depend on text editors to "debug" their document if the word processor makes a mistake, or allows the user to make a mistake. The only way to allow authors convenient access to both the structure and data of the document in standard text editors is to put the two right beside each other, "cheek to cheek".

As we discussed in the introduction, the stuff that represents the logical structure and connects the entities is called markup. An XML document is made up exclusively of markup and character data. Both are in Unicode. Both are termed *XML text.* This last point is important! No matter how intuitive it might seem, we do not use the word "text" to mean character data.

> **Caution** *The term XML text refers to the combination of character data and markup, not character data alone. Character data + markup = text.*

Markup is differentiated from character data by special characters called *delimiters*. Informally, text between a less-than ("<") and a greater-than (">") character or between an ampersand ("&") and a semicolon (";") character is markup. Those four characters are the most common delimiters. This rule will become more concrete in later chapters. In the meantime, here is an example of a small document to give you a taste of XML markup.

The markup between the less-than and greater-than is called a *tag*.

You may be familiar with other languages that use similar syntax. These include HTML and other SGML-based languages.

3.6 | Document types

The concept of a document type is fairly intuitive. You are well aware that letters, novels and telephone books are quite different, and you are probably comfortable recognizing documents that conform to one of these categories. No matter what its title or binding, you would call a book that listed

Example 3-1. A small XML document

```
<?xml version="1.0"?>
<!DOCTYPE Q-AND-A SYSTEM "http://www.q.and.a.com/faq.dtd">
<Q-AND-A>
<QUESTION>I'm having trouble loading a WurdWriter 2.0 file into
WurdPurformertWriter 7.0. Any suggestions?</QUESTION>

<ANSWER>Why don't you use XML?</ANSWER>

<QUESTION>What's XML?</QUESTION>

<ANSWER>It's a long story, but there is a book I can
recommend...</ANSWER>
</Q-AND-A>
```

names and phone numbers a phone book. So, a document type is defined by its elements. If two documents have radically different elements or allow elements to be combined in very different ways then they probably do not conform to the same document type.

This notion of a document type can be formalized in XML. A *document type definition* (or *DTD*) is a series of definitions for element types, attributes, entities and notations. It declares which of these are legal within the document and in what places they are legal. A document can claim to conform to a particular DTD in its *document type declaration.*[1]

DTDs are powerful tools for organizational standardization in much the same way that forms, templates and style-guides are. A very rigid DTD that only allows one element type in a particular place is like a form: "Just fill in the blanks!". A more flexible DTD is like a style-guide in that it can, for instance, require every `list` to have two or more `items`, every `report` to have an `abstract` and could restrict `footnotes` from appearing within `footnotes`.

DTDs are critical for organizational standardization, but they are just as important for allowing robust processing of documents by software. For example, a letter document with a `chapter` in the middle of it would be most unexpected and unlikely to be very useful. Letter printing software would not reliably be able to print such a document because it is not well defined what a chapter in a letter looks like. Even worse is a situation where a document is missing an element expected by the software that processes

[1] The document type declaration is usually abbreviated "DOCTYPE", because the obvious abbreviation would be the same as that for document type definition!

it. If your mail program used XML as its storage format, you might expect it to be able to search all of the incoming email addresses for a particular person's address. Let us presume that each message stores this address in a `from` element. What do we do about letters without `from` elements when we are searching them? Programmers could write special code to "work around" the problem, but these kinds of workarounds make code difficult to write.

HTML serves as a useful cautionary tale. It actually has a fairly rigorous structure, defined in SGML, and available from the World Wide Web Consortium. But everybody tends to treat the rules as if they actually came from the World Wrestling Federation – they ignore them.

The programmers that maintain HTML browsers spend a huge amount of time incorporating support for all of the incorrect ways people combine the HTML elements in their documents. Although HTML has an SGML DTD, very few people use it, and the browser vendors have unofficially sanctioned the practice of ignoring it. Programming workarounds is expensive, time consuming, boring and frustrating, but the worst problem is that there is no good definition of what these illegal constructs mean. Some incorrect constructs will actually make HTML browsers crash, but others will merely make them display confusing or random results.

In HTML, the `title` element is used to display the document's name at the top of the browser window (on the title bar). But what should a browser do if there are two titles? Use the first? Use the last? Use both? Pick one at random? Since the HTML standard does not allow this construct it certainly does not specify a behavior. Believe it or not, an early version of Netscape's browser showed each title sequentially over time, creating a primitive sort of text animation. That behavior disappeared quickly when Netscape realized that authors were actually creating invalid HTML specifically to get this effect! Since authors cannot depend on non-sensical documents to work across browsers, or even across browser versions, there must be a formal definition of a valid, reasonable document of a particular type. In XML, the DTD provides a formal definition of the element types, attributes and entities allowed in a document of a specified type.

There is also a more subtle, related issue. If you do not stop and think carefully about the structure of your documents, you may accidently slip back into specifying them in terms of their formatting rather than their abstract structure. We are accustomed to thinking of documents in terms of their rendition. That is because, prior to GML, there was no practical way to create a document without creating a rendition. The process of creating a

DTD gives us an opportunity to rethink our documents in terms of their structure, as abstractions.

Here are examples of some of the declarations that are used to express a DTD:

> **Caution** *A DTD is a concept; markup declarations are the means of expressing it. The distinction is important because other means of expressing DTDs are being proposed (see Chapter 39, "XML-Data", on page 570). However, most people, even ourselves, don't make the distinction in normal parlance. We just talk about the declarations as though they are the DTD that they describe.*

Example 3-2. Markup declarations

```
<!ELEMENT Q-AND-A (QUESTION,ANSWER)+>
<!-- This allows: question, answer, question, answer ... -->

<!ELEMENT QUESTION (#PCDATA)+>
<!-- Questions are just made up of text -->

<!ELEMENT ANSWER (#PCDATA)+>
<!-- Answers are just made up of text -->
```

Some XML documents do not have a document type declaration. That does not mean that they do not conform to a document type. It merely means that they do not claim to conform to some formally defined document type definition.

If the document is to be useful as an XML document, it must still have some structure, expressed through elements, attributes and so forth. When you create a stylesheet for a document you will depend on it having certain elements, on the element type names having certain meanings, and on the elements appearing in certain places. However it manifests itself, that set of things that you depend on is the document type.

You can formalize that structure in a DTD. In addition to or instead of a formal computer-readable DTD, you can also write out a prose description. You might consider the many HTML books in existence to be prose definitions of HTML. Finally, you can just keep the document type in your head and maintain conformance through careful discipline. If you can achieve this for large, complex documents, your powers of concentration are astounding! Which is our way of saying: we do not advise it. We will discuss DTDs more in Chapter 32, “Creating a document type definition”, on page 448.

3.7 | Well-formedness and validity

Every language has rules about what is or is not valid in the language. In human languages that takes many forms: words have a particular correct pronunciation (or range of pronunciations) and they can be combined in certain ways to make valid sentences (grammar). Similarly XML has two different notions of “correct”. The first is merely that the markup is intelligible: the XML equivalent of “getting the pronunciation right”. A document with intelligible markup is called a *well-formed* document. One important goal of XML was that these basic rules should be simple so that they could be strictly adhered to.

The experience of the HTML market provided a cautionary tale that guided the development of XML. Much of the HTML on the Web does *not* conform to even the simplest rules in the HTML specifications. This makes automated processing of HTML quite difficult.

Because Web browsers will display ill-formed documents, authors continue to create them. In designing XML, we decided that XML processors should actually be prohibited from trying to recover from a *well-formedness* error in an XML document. This was a controversial decision because there were many who felt that it was inappropriate to restrict XML implementors from deciding the best error recovery policy for their application.

The XML equivalent of “using the right words in the right place” is called *validity* and is related to the notion of document types. A document is *valid* if it declares conformance to a DTD in a document type declaration and actually conforms to the DTD.

Documents that do not have a document type declaration are not really *invalid* – they do not violate their DTD – but they are not valid either, because they cannot be validated against a DTD.

If HTML documents with multiple titles were changed over to use XML syntax, they would be *well-formed* and invalid (presuming the HTML DTD was also converted to XML syntax). If we remove the document type declaration, so that they no longer claim to conform to the HTML DTD, then they would become merely well-formed but neither valid nor invalid.

> ***Caution*** *For most of us, the word "invalid" means something that breaks the rules. It is an easy jump from there to concluding that an XML document that does not conform to a DTD is free to break any rules at all. So for clarity, we may sometimes say "type-valid" and "non-type-valid", rather than "valid" and "invalid".*

You should think carefully before you decide to make a document that is well-formed but not valid. If the document is one-of-a-kind and is small, then making it well-formed is probably sufficient. But if it is to be part of any kind of information system (even a small one) or if it is a large document, then you should write a DTD for it and validate your document regularly. When you decide to build or extend your information system, the fact that the document is guaranteed to be consistent will make your programming or stylesheet writing many times easier and your results much more reliable.

3.8 | Hyperlinking and Addressing

If you have used the Web, then you probably do not need to be convinced of the importance of hyperlinking. One thing you might not know, however, is that the Web's notions of hyperlink are fairly tame compared to what is available in the best academic and commercial hypertext systems. XML alone does not correct this, but it has an associated standard called XLink that goes a long way towards making the Web a more advanced hypertext environment.

The first deficiency of today's Web links is that there are no standardized mechanisms for making links that are external to the documents that they are linking from. Let's imagine, for example that you stumble upon a Web page for your favorite music group. You read it, enjoy it and move on. Imagine next week you stumble upon a Web page with all of the lyrics for all of their songs (with appropriate copyrights, of course!). You think: there should be a link between these two pages. Someone visiting one might want to know about the other and vice versa.

What you want to do is make an *external link*. You want to make a link on your computer that appears on both of the other computers. But of course you do not have the ability to edit those two documents. XLink will allow this external linking. It provides a representation for external links, but it does not provide the technology to automatically publish those links to the world. That would take some kind of *link database* that would track all of the links from people around the world. Needless to say this is a big job and though there are prototypes, there is no standardized system yet.

You may wonder how all of these links will be displayed, how readers will select link sheets and annotations, how browsers will talk to databases and so forth. The simple answer is: “nobody knows yet.”[1]

Before the first Web browser was developed there was no way to know that we would develop a convention of using colored, underlined text to represent links (and even today some browsers use other conventions). There was also no way to know that browsers would typically have “back” buttons and “history lists”. These are just conventions that arose and browser features that became popular.

This same process will now occur with external links. Some user interface (perhaps a menu) will be provided to apply external link sheets, and there will probably be some mechanism for searching for link sheets related to a document on the Web. Eventually these will stabilize into standards that will be ubiquitous and transparent (we hope!). In the meantime, things are confused, but that is the price for living on the cutting edge. XLink moves us a notch further ahead by providing a notation for representing the links.

Another interesting feature of XML extended links is that they can point to more than one resource. For instance instead of making a link from a word to its definition, you might choose to link to definitions in several different dictionaries. The browser might represent this as a popup menu, a tiny window with the choices listed, or might even open one window for each. The same disclaimer applies: the XML Link specification does not tell browsers exactly what they must do. Each is free to try to make the most intuitive, powerful user interface for links. XML brings many interesting hypertext ideas from university research labs and high tech companies “to the masses.” We still have to work out exactly how that will look and who will use them for what. We live in interesting times!

[1] But we've got some ideas. See Chapter 13, “Extended linking”, on page 176.

3.9 | Stylesheets

To a certain extent, the concerns described above are endemic to generalized markup. Because it describes structure, and not formatting, it allows variations in display and processing that can sometimes disturb people.

However, as the Web has evolved, people have become less and less tolerant of having browser vendors control the "look and feel" of their documents. An important part of all communication, but especially modern business communication, is the idea of style. Stylesheets allow us to attach our own visual style to documents without destroying the virtue of generalized markup. Because the style is described in a separate entity, the stylesheet, software that is not interested in style can ignore it.

For instance most search engines would not care if your corporate color is blue or green, so they will just ignore those declarations in the stylesheet. Similarly, software that reads documents aloud to the sight-impaired would ignore font sizes and colors and concentrate on the abstractions – paragraphs, sections, titles and so forth.

The Web has a very simple stylesheet language called *Cascading Style Sheets* (CSS), which arose out of the early battles between formatting and generalized markup in HTML. Like any other specification, CSS is a product of its environment, and so is not powerful enough to describe the formatting of documents types that are radically different in structure from HTML.

Because CSS is not sufficient, the World Wide Web Consortium is working on a complementary alternative called the Extensible Stylesheet Language (XSL). XSL will have many features from CSS, but will also borrow some major ideas from ISO's DSSSL stylesheet language. XSL will be extensible, just as XML is, so that it will be appropriate for all document types and not just for HTML. Like the linking specification, XSL is still under development so its exact shape is not known. Nevertheless, there is a proposal for a general design that we will review later on.

3.10 | Conclusion

There are a lot of new ideas here to absorb, but we'll be repeating and reemphasizing them as we move along. At this point, though, we're ready to look in-depth at the ways that XML is being used in the real world.

XML in the real world

- Real-world concepts
- Application scenarios
- Case studies
- XML tools
- Jargon demystified

Chapter 4

Applications are the reason for using technology, so it makes sense to get a good idea of what XML is used for before digging into the details of the language.

And since XML may be somewhat different from the technologies that you are accustomed to using, it is also helpful to see how people actually work with it; how the tools are used.

We're going to cover those subjects at length in the next three parts of the book. In preparation, we need to examine some often elusive – but vital – concepts relating to real-world use of XML.

4.1 | Is XML for documents or for data?

What is a document?

My dictionary says:

> "Something written, inscribed, engraved, etc., which provides evidence or information or serves as a record".

Documents come in all shapes and sizes and media, as you can see in Figure 4-1. Here are some you may have encountered:

- Long documents: books, manuals, product specifications
- Broadsides: catalog sheets, posters, notices
- Forms: registration, application, etc.
- Letters: email, memos
- Records: "Acme Co., Part# 732, reverse widget, $32.50, 5323 in stock"
- Messages: "job complete", "update accepted"

An e-commerce transaction, such as a purchase, might involve several of these. A buyer could start by sending several documents to a vendor:

- Covering note: a letter
- Purchase order: a form
- Attached product specification: a long document

The vendor might respond with several more documents:

- Formal acknowledgment: a message
- Thank you note: a letter
- Invoice: a form

The beauty of XML is that the same software can process all of this diversity. Whatever you can do with one kind of document you can do with all the others. The only time you need additional tools is when you want to do different kinds of things – not when you want to work with different kinds of documents.

And there are lots of things that you can do.

4.2 | Endless spectrum of application opportunities

Sorry about that, we've been reading too many marketing brochures. But it's true, nevertheless.

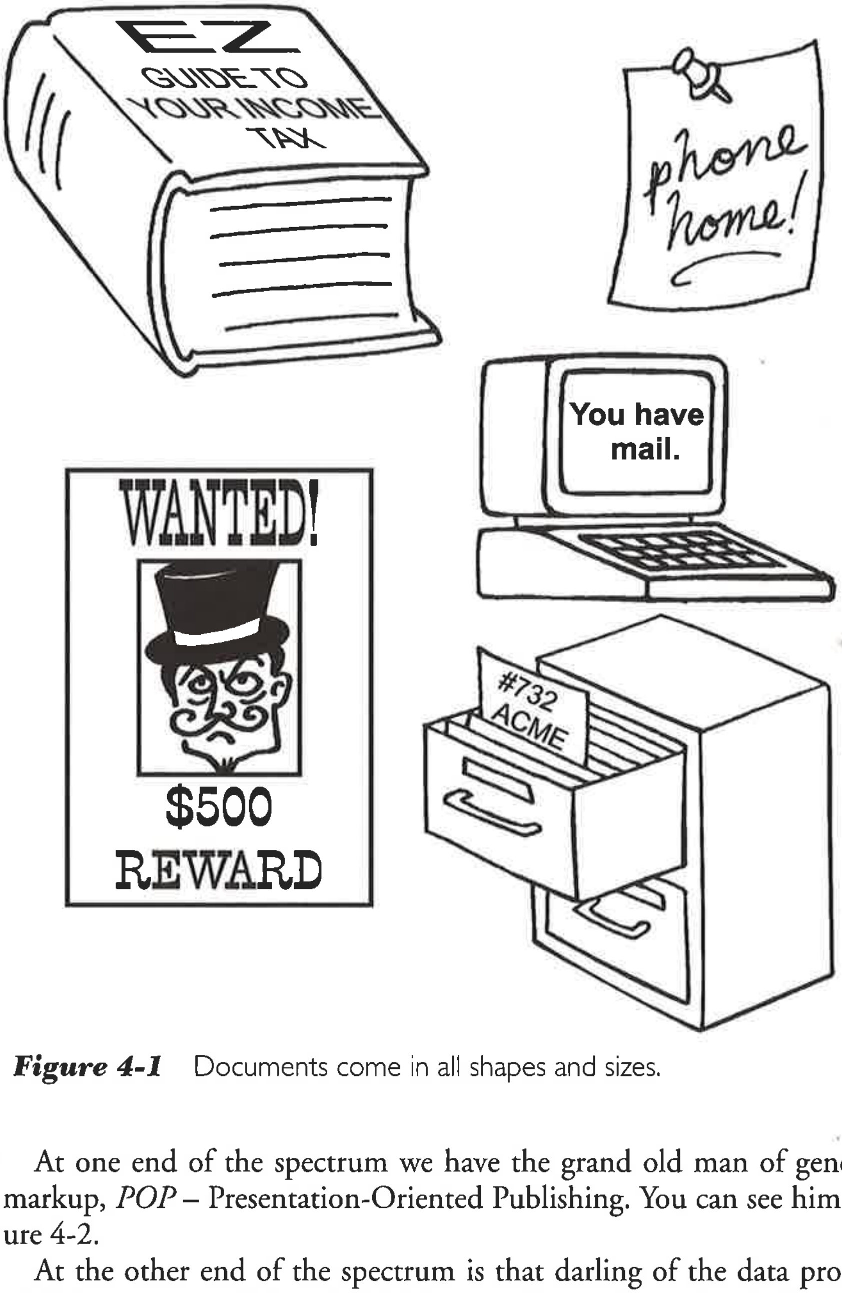


Figure 4-1 Documents come in all shapes and sizes.

At one end of the spectrum we have the grand old man of generalized markup, *POP* – Presentation-Oriented Publishing. You can see him in Figure 4-2.

At the other end of the spectrum is that darling of the data processors, *MOM* – Message-Oriented Middleware. She smiles radiantly from Figure 4-3.

Let's take a closer look at both of them.

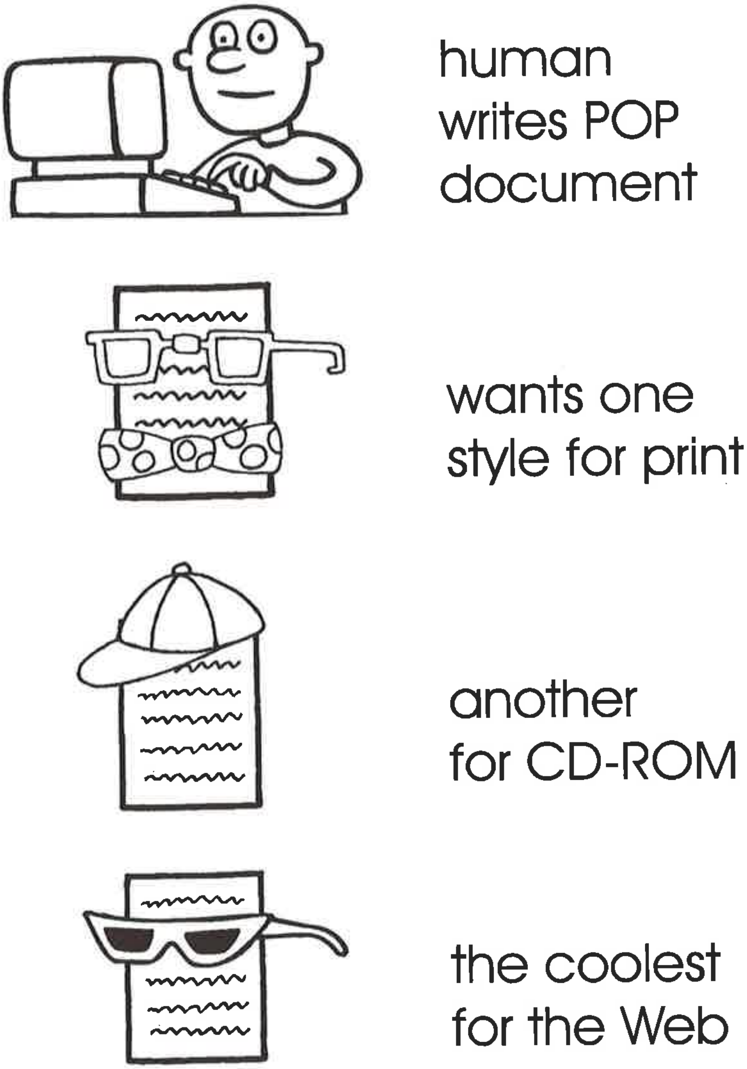


Figure 4-2 POP application.

4.2.1 *Presentation-oriented publishing*

POP was the original killer app for SGML, XML's parent, because it saves so much money for enterprises with Web-sized document collections.

POP documents are chiefly written by humans for other humans to read.

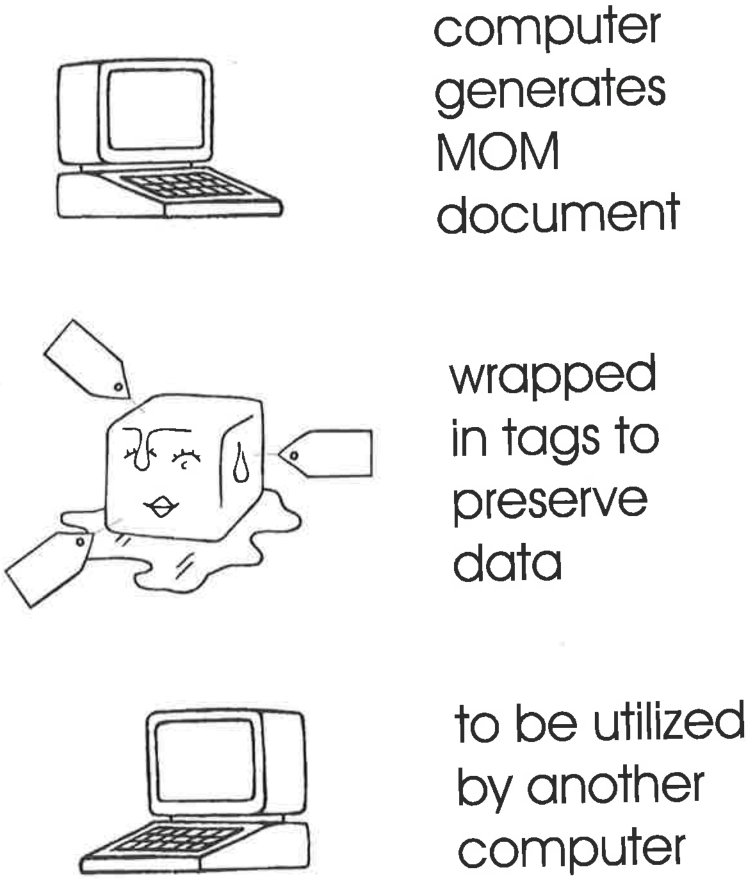

computer generates MOM document
wrapped in tags to preserve data
to be utilized by another computer

Figure 4-3 MOM application.

Instead of creating formatted renditions, as in word processors or desktop publishing programs, XML POP users create unformatted abstractions. That means the document file captures what is *in* the document, but not how it is supposed to look.

To get the desired look, the POP user creates a stylesheet, a set of commands that tell a program how to format (and/or otherwise process) the document. The power of XML in this regard is that you don't need to choose just one look – you can have a separate stylesheet for every purpose.[1]

[1] We know that all office suites have some degree of stylesheet support today, but XML (well, GML) did it first, and still is the only way to do it cleanly.

At a minimum, you might want one for print, one for CD-ROM, and another for a Web site.

POP documents tend to be (but needn't be) long-lived, large, and with complex structures. When delivered in electronic media, they may be interactive. How they will be rendered is of great importance, but, because XML is used, the rendition information can be – and is – kept distinct from the abstract data.

4.2.2 *Message-oriented middleware*

MOM is the killer app – actually, a technology that drives lots of killer apps – for XML on the Web.

Middleware, as you might suspect from the name, is software that comes between two other programs. It acts like your interpreter/guide might if you were to visit someplace where you couldn't speak the language and had no idea of the local customs. It talks in the native tongue, using the native customs, and translates the native replies – the messages – into your language.

MOM documents are chiefly generated by programs for other programs to read.

Instead of writing specialized programs (clients) to access particular databases or other data sources (servers), XML MOM users break the old two-tier client/server model. They introduce a third tier, the "middle tier", that acts as a data integrator. The middle-tier server does all the talking to the data sources and sends their messages in XML to the client.

That means the client can read data from anywhere, but only has to understand data that is in XML documents. The XML markup provides information about the data (i.e., metadata) that was in the original data source schema, like the database table name and field names (also called "cell" or "column" names).

The MOM user typically doesn't care much about rendition. He does care, though, about extracting the original data accurately and making some use of the metadata. His client software, instead of having a specialized module for each data source, has a single "XML parser" module. The parser is the program that separates the markup from the data, just as it does in POP applications.

And just like POP applications, there can be a stylesheet, a set of commands that tell a program how to process the document. It may not look much like a POP stylesheet – it might look more like a script or program – but it performs the same function. And, as with POP stylesheets, there can be different MOM stylesheets for different document types, or to do different things with message documents of a single document type.

There is an extra benefit to XML three-tier MOM applications in a networked environment. For many applications, the middle-tier server can collect all of the relevant data at once and send it in a single document to the client. Further querying, sorting, and other processing can then take place solely on the client system. That not only cuts down Web traffic and overhead, but it vastly improves the end-user's perceived performance and his satisfaction with the experience.

MOM documents tend to be (but needn't be) short-lived, non-interactive, small, and with simple structures.

4.2.3 *Opposites are attracted*

To XML, that is!

How is it that XML can be optimal for two such apparently extreme opposites as MOM and POP? The answer is, the two are not really different where it counts.

In both cases, we start with abstract information. For POP, it comes from a human author's head. For MOM, it comes from a database. But either way, the abstract data is marked up with tags and becomes a document.

Here is a terminally cute mnemonic for this very important relationship:

Data + Markup = DocuMent

Aren't you sorry you read it? Now you'll never forget it.

But XML "DocuMents" are special. An application can do three kinds of processing with one:

- *Parse it*, in order to extract the original data. This can be done without information loss because XML represents both metadata and data, and it lets you keep the abstractions distinct from rendition information.
- *Render it*, so it can be presented in a physical medium that a human can perceive. It can be rendered in many different ways, for delivery in multiple media such as screen displays, print, Braille, spoken word, and so on.

- *Hack it*, meaning "process it as plain text without parsing". Hacking might involve cutting and pasting into other XML documents, or scanning the markup to get some information from it without doing a real parse.

The real revelation here is that data and documents aren't opposites. Far from it – they are actually two states of the same information.

The real difference between the two is that when data is in a database, the metadata about its structure and meaning (the schema) is stored according to the proprietary architecture of the database. When the data becomes a document, the metadata is stored as markup.

A mixture of markup and data must be governed by the rules of some *notation*. XML and SGML are notations, as are RTF and Word file format. The rules of the notation determine how a parser will interpret the document text to separate the data from the markup.

Notations are not just for complete documents. There are also data object notations, such as GIF, TIFF, and EPS, that are used to represent such things as graphics, video (e.g., MPEG), and audio (e.g., AVI). Document notations usually allow their documents to contain data objects, such as pictures, that are in the objects' own data object notations.

Data object notations are usually (not always) in *binary*; that is, they are built-up from low-level ones and zeros. Document notations, however, are frequently *character-based*. XML is character-based, which is why it can be hacked.

In fact, a design objective of XML was to support the "desperate Perl hacker" – someone who needs to write a program in a hurry, using a scripting language like Perl, and who doesn't use a real XML parser. Instead, his program scans the XML document as though it were plain text. The program might search for markup strings, but can also search for data.

A hacker[1] often uses cues that have special meaning to him, like giving special treatment to a tag that occurs at the start of a line, even though those cues have no meaning to a parser. That's why serious hackers do their XML editing with programs that can preserve a document's source and

[1] As used here, and by most knowledgeable computer people, "hacker" has none of the "cracker" stigma given the term in the popular press. The only security compromised by a desperate Perl hacker is his job security, for leaving things to the last minute!

reproduce it character-for-character. They don't let the software decide which characters are important enough to preserve.

Since databases and documents are really the same, and MOM and POP applications both use XML documents, there are lots of opportunities for synergy.

4.2.4 *MOM and POP – They're so great together!*

Classically, MOM and POP were radically different kinds of applications, each doing things its own way with different technologies and mental models. But POP applications frequently need to include database data in their document content – think of an automotive maintenance manual that has to get the accurate part numbers from a database.

Similarly, MOM applications need to include human-written components. When the dealer asks for price and availability of the automotive parts you need, the display might include a description as well.

With the advent of generalized markup, the barriers to doing MOM-like things in POP applications began to disappear. Some of the POP-like applications you'll read about in the next part of the book appear to have invented the middle tier on their own. And now, with the advent of XML, MOM applications can easily incorporate POP functionality as well.

In fact, we'd go so far as to say there is no longer a difference in kind between the two, only a difference in degree. There really is "an endless spectrum of application opportunities". It is a multi-dimensional spectrum where applications need not be implemented differently just because they process different document types. The real differentiators are other document characteristics, like persistency, size, interactivity, structural complexity, percentage of human-written content, and the importance of eventual presentation to humans.

At the extremes, some applications may call for specialized (or optimized) techniques, but the broad central universe of applications can all be implemented similarly. Much of the knowledge that POP application developers have acquired over the years is now applicable to MOM applications, and vice versa. Keep that in mind as you read the application descriptions and case studies.

That cross-fertilization is true of products and their underlying technologies as well. All of the product descriptions in this book should be of interest, whether you think of your applications as chiefly being MOM or being POP. It is the differences in functionality and design that should cause you to choose one product over another, not their marketing thrust or apparent orientation. We've included detailed usage examples for leading tools in each category so you can look beyond the labels.

4.3 | XML tools

Our coverage of tools falls into three broad categories.

Editing and composition

These are the classic tools of POP applications, but now with applicability to the MOM world as well. Editors are used for creating and revising documents. Composition tools produce renditions, but composition functionality is sometimes included in editors.

Content management

A major benefit of XML is the ability to store and work with components of documents, rather than only being able to deal with the document as a whole. These tools use databases to store information components so they can be controlled, managed, and assembled into end-products in the same way as components of automobiles, aircraft, or other complex devices. Think of them as the MOM and POP store (Figure 4-4).[1]

Middle-tier tools

These are the vital MOM application tools for creating middle-tier servers. They integrate data sources and allow applications to interoperate.

[1] Generations ago the Mom and Pop store (grocery, convenience, etc.) was the achievement of the entrepreneurial couple who'd lifted themselves out of the working class. Today they'd have an Internet start-up and be striving for a successful IPO!

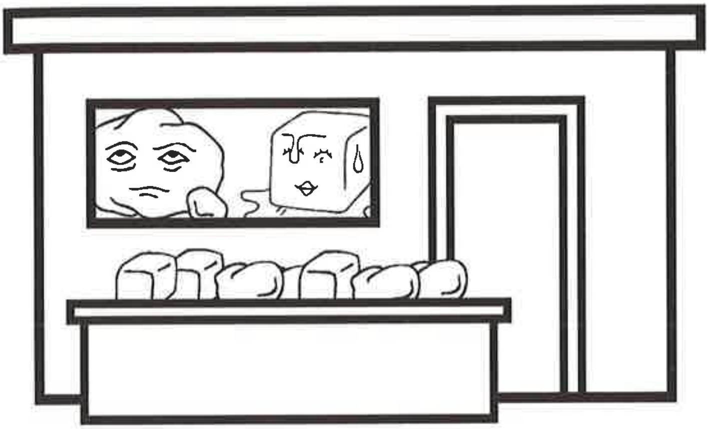


Figure 4-4 Content management: The MOM and POP store.

In each category, we cover a number of products with detailed usage examples. Although there is often functional overlap among them, each has unique strengths that are targeted towards a particular kind of use. We've tried to emphasize those differences in order to discuss different tool characteristics in each chapter.

There is also a survey of tools that are available for free, in categories such as XML parsers, XSL engines, converters, and viewers. Some 55 of them are on the CD-ROM accompanying this book.

Tool capabilities are also discussed in the application scenarios and case studies.

4.4 | XML jargon demystifier

One of the problems in learning a new technology like XML is getting used to the jargon. A good book will hold you by the hand, introduce terms gradually, and use them precisely and consistently.

Out in the real word, though, people use imprecise terminology that often makes it hard to understand things, let alone compare products. And, unlike authors,[1] they sometimes just plain get things wrong.

[1] We should be so lucky!

For example, you may see statements like "XML documents are either well-formed or valid." As you've learned from this book, that simply isn't true. All XML documents are well-formed; some of them are also valid.[1]

In this book, we've taken pains to edit the application and tool chapters to use consistent and accurate terminology. However, for product literature and other documents you read, the mileage may vary. So we've prepared a handy guide to the important XML jargon, both right and wrong. Think of it as a MOM application for XML knowledge.

4.4.1 *Structured vs. unstructured*

XML documents are frequently referred to as *structured* while other text, such as rendition notations like RTF, are called *unstructured.*

In fact, renditions can have a rich structure, composed of elements like pages, columns, and blocks. The real distinction being made is between "abstract" and "rendered". Keep that in mind when you read about "structured" and "unstructured", even in this book

4.4.2 *Tag vs. element*

Tags aren't the same thing as elements. Tags describe elements.

In Figure 4-5 the package, metaphorically speaking, is an element. The contents of the package is the content of an element. The tag describes the element. It contains two names:

- the *element type name* ("Wristwatch"), which says what type of element it is, and
- a *unique identifier*, or ID ("WW42-3729"), which says which particular element it is.

A tag could also include attributes describing other properties of the element, such as *Manufacturer="Hy TimePiece Company"*.

When people talk about a *tag name*:

[1] So does that mean a merely well-formed document is "invalid"? No, an invalid document is one that isn't well-formed; it breaks the rules of the XML notation. Hey, we didn't promise to justify XML jargon, just to explain it.

1.They are referring to the element type name.

2.They are making an error, because tags aren't named.

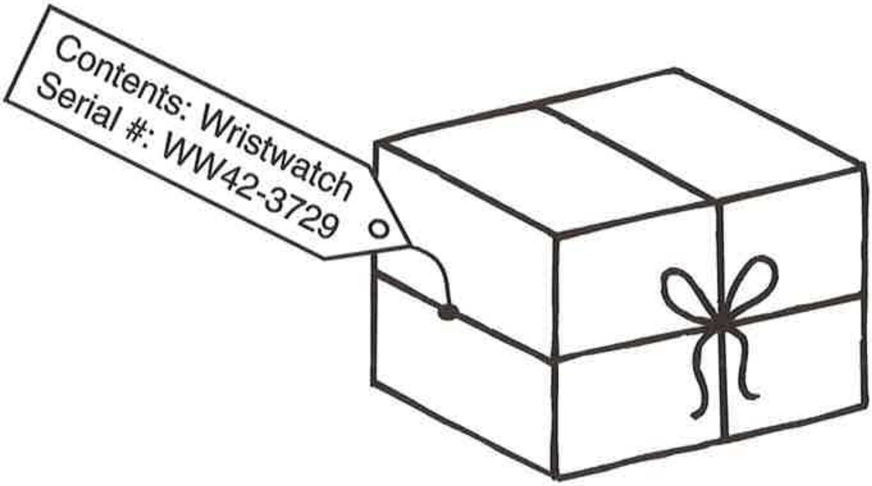


Figure 4-5 What's in a tag?

4.4.3 *Document type, DTD, and markup declarations*

A *document type* is a class of similar documents, like telephone books, technical manuals, or (when they are marked up as XML) inventory records.

A *document type definition* (*DTD*) is the set of rules for using XML to represent documents of a particular type. These rules might exist only in your mind as you create a document, or they may be written out.

Markup declarations, such as those in Example 4-1, are XML's way of writing out DTDs.

Example 4-1. Markup declarations in the file `greeting.dtd`**.**

```
<!ELEMENT greeting (salutation, addressee) >
<!ELEMENT salutation (#PCDATA) >
<!ELEMENT addressee  (#PCDATA) >
```

It is easy to mix up these three constructs: a document type, XML's markup rules for documents of that type (the DTD), and the expression of those rules (the markup declarations). It is necessary to keep the constructs separate if you are dealing with two or more of them at the same time, as when discussing alternative ways to express a DTD. But most of the time, even in this book, "DTD" will suffice for referring to any of the three.

4.4.4 *Document, XML document, and document instance*

The term *document* has two distinct meanings in XML.

Consider a really short XML document that might be rendered as:

Hello World

In one sense, the abstract message you get in your mind when you read the rendition is the *real document.* Communicating that abstraction is the reason for using XML in the first place.

In a formal, syntactic sense, though, the complete text (markup + data, remember) of Example 4-2, is the *XML document.* Perhaps surprisingly, that includes the markup declarations for its DTD in Example 4-1. The XML document, in other words, is a character string that *represents* the real document.

In this example, much of that string consists of the markup declarations, which express the greeting DTD. Only the last four lines describe the real document, which is an instance of a greeting. Those lines are called the *document instance.*

Example 4-2. A greeting document.

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "file://greeting.dtd">
<greeting>
<salutation>Hello</salutation>
<addressee>World</addressee>
</greeting>
```

4.4.5 *Coding, encoding, and markup*

People refer to computer programs as *code*, and to the act of programming as *coding*.

There is also the word *encoding*, which refers to the way that characters are represented as ones and zeros in computer storage. XML has a declaration for specifying an encoding.

You'll often see (in places other than this book) phrases like "XML-encoded data", "coded in HTML", or "XML coding".

But using XML isn't coding. Not in the sense of programming, and not in the sense of character encoding. What those phrases mean are "XML document", "marked-up in HTML", and "XML markup".[1]

4.5 | Conclusion

We've covered the key concepts of XML itself, and of the ways in which it is used in the real world. Now we are ready to examine those real-world uses in depth, with application scenarios, case studies of actual users, and detailed descriptions of the tools of the tag trade.

[1] Although dynamic HTML pages contain so much scripting that the phrase "HTML coding" is almost warranted.

Building an online auction Web site

- Three-tier Web application
- Dynamic generation of XML documents
- Extracting data from XML documents
- Creating a user interface

Personalized frequent-flyer Web site

- Three-tier XML Web application
- What makes Web sites "hot"
- Client/server Web model is changing
- Website personalization

Chapter 5

"If the current frequent-flyer Web site model is the ultimate in doing business on the Web, I predict that business on the Web will never really take off". So says high-flying XML consultant Dianne Kennedy of XMLXperts Ltd., `http://www.xmlxperts.com`, who prepared this chapter. It is sponsored by SoftQuad Inc., `http://www.sq.com`.

If you surf the Web as well as travel by air, you have probably stopped by your favorite airline frequent-flyer Web site. How would you rate that experience?

5.1 | Today's frequent-flyer sites

It might have been fun to find the site and to see all the last minute "bargains" offered for frequent flyers. Perhaps those specials were initially enough to motivate you to return to the site, if only to dream of taking a vacation in the middle of your biggest project!

Beyond viewing the posted specials, perhaps you interacted with the site in a limited way, by entering your frequent flyer number to see your current point balance. But during heavy traffic hours on the Web, such interactions can take quite a long time.

And once you know how many points you've accumulated, what about the whole series of new questions it stimulates for which the Web site can't provide an answer. At that point, you must resort to calling the "1-800" number to learn more about your award options and eventually book a flight.

Bottom line: once the novelty wears off, this Web experience, like countless others, is less than satisfying.

5.2 | What's wrong with today's Web model?

Today's Web model is a "client/server" model. In this model, any personalized interaction takes place on the Web server you have contacted. As a result, there is little of it.

Typical Web pages today are static brochures rendered in HTML to provide eye-appealing display. In fact, the Web sites that are rated the "hottest" in today's market are those that provide multi-media sizzle – heavy on graphics, animation, and sound. Personalized content, while a consideration, has not yet become the primary distinguishing characteristic of a "hot" Web site. But as the shift takes place from simply providing entertainment value to facilitating business transactions, personalized content will become "hot".

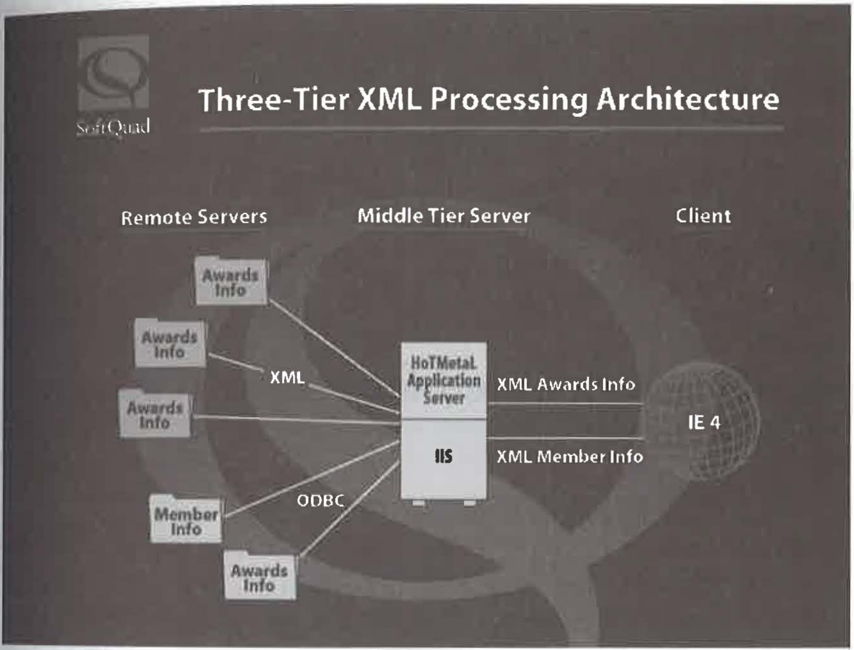
In today's airline frequent-flyer Web sites, there is a great deal of HTML information that the customer can view. If this information and its associated links changes daily, the Web site becomes more interesting and is more likely to generate return visits. Likewise, interactivity generates more site traffic.

But currently interactivity requires lengthy periods where the customer must be "connected" to remote servers. Queries from the customer go to the server, and resulting responses are shipped back to the customer for viewing in HTML. Unfortunately, a Web server can handle only a limited number of connections at one time.

Every time a new piece of information is requested, a transaction between the client's Web browser and remote Web servers is required. Sooner or later the number of transactions slows the server and the customer experiences lengthy time-outs when queries are processed and data is transferred back to the browser.

5.3 | A better model for doing business on the Web

Today, XML has enabled a new breed of Web server software, one that allows the Web developer to add a new "middle tier" server to the Web model. One example of such software is *HoTMetaL Application Server*, described in Chapter 28, "HoTMetaL Application Server", on page 378. It is used in Figure 5-1 to illustrate the new three-tier Web architecture.



SoftQuad

Three-Tier XML Processing Architecture

Remote Servers

Middle Tier Server

Client

Figure 5-1 Three-tier architecture with *HoTMetaL Application Server.*

Remember, in the old Web model, the customer using a browser such as *Internet Explorer* or *Netscape* on the client interacted directly with data sources on remote servers. The client maintained its connection throughout the interactive session. Each query was sent a response in HTML which could be directly viewed by the client browser. Maintaining the connection between the client and server was critical.

In the new three-tier Web model, the information that fits the profile of the customer is retrieved at once from remote databases by software on the middle tier, either as XML documents or through an ODBC or similar database connection. From that point, continued interaction with the remote databases is no longer required. The connection to the remote servers can be, and is, terminated.

Once all information that fits the customer profile has been assembled by software on the middle tier, it is sent in XML to the client. Now the requirement for further interaction between the client and the middle tier server is eliminated as well.

Rich XML data, directly usable by client applications and scripting languages like *JavaScript*, has been delivered to the client. The connection between the client and middle tier server can now be terminated. At this point, all computing becomes client-based, resulting in a much more efficient use of the Web and a much more satisfying customer experience.

To understand the new three-tier Web model better, one must understand the role XML plays as an enabling technology. One must also understand how efficient delivery of structured data to the client makes all the difference.

5.4 | An XML-enabled frequent-flyer Web site

Initially, differences between the Softland Air XML-enabled frequent-flyer Web site shown in Figure 5-2 and existing frequent-flyer sites may not be apparent. Both provide a pleasing HTML-rendered site brochure. Both enable you to select the services you wish to use. But here the similarities end. New business functions, not possible with today's non-XML sites, quickly become apparent.

From the initial *Softland Air* screen you can select the "frequent flyer" option. This option will cause the frequent-flyer page in Figure 5-3 to be displayed, by traversing a simple hyperlink. When the frequent-flyer page is displayed, it asks you to enter your membership identification number.

Once you have entered your membership number, a personalized, interactive Web experience begins. The next screen that is displayed (Figure 5-4) not only returns your number of frequent flyer points, but shows you destinations for which you have already qualified for awards. This screen will

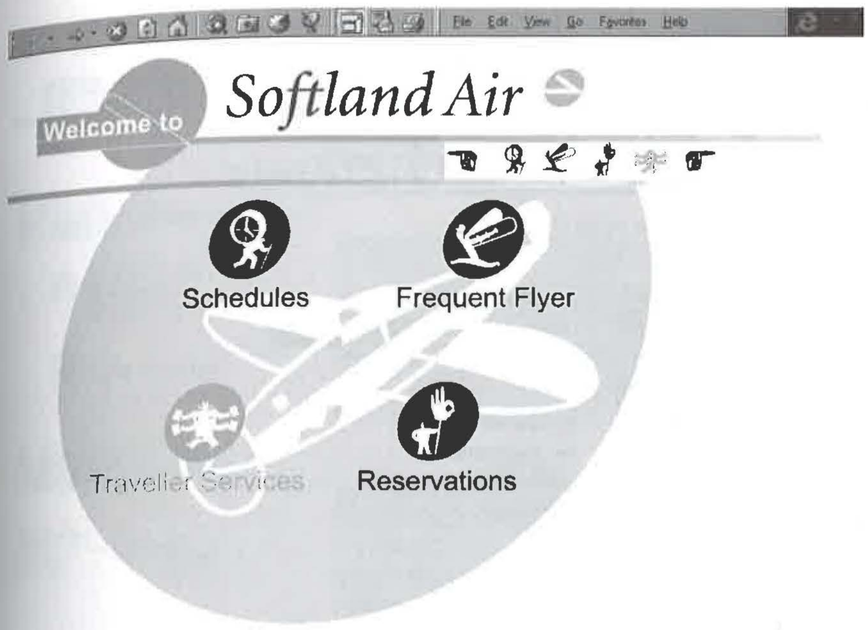


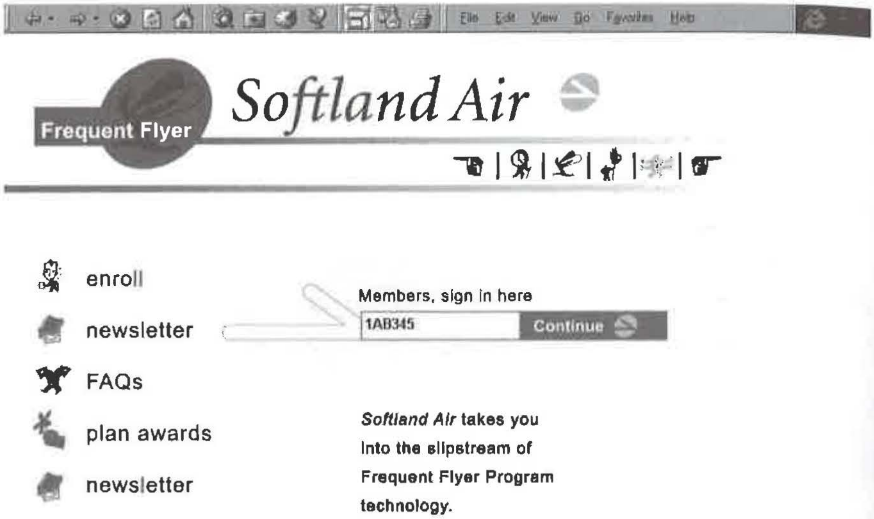
Figure 5-2 "Welcome to *Softland Air*"

vary from member to member, based upon the points a member has in the frequent-flyer database and other personal information the database holds, such as point of origin.

In addition to showing you the awards you have already earned, the *Softland Air* Web site enables you to select destinations of interest. You can see that you have 46,000 points and are qualified to go anywhere in North America in economy through first class. You can also go to Hawaii or the Caribbean by economy class. You do not qualify to go to Europe, but you can see that you nearly have enough points for a European trip.

Suppose you are interested in going to Europe. To learn more about options to get there, you would select a destination on the "Awards Specials" part of the screen. This destination information is added to your profile, along with your point of origin and the number of points you currently have. It will be used to personalize the ongoing transactions.

Once you have selected a destination, the Web page shows you awards, both on *Softland Air* and on partner airlines, that fit the destination you have selected. From this screen you can see what destinations in Europe



Members, sign in here

1AB345

Softland Air takes you into the slipstream of Frequent Flyer Program technology.

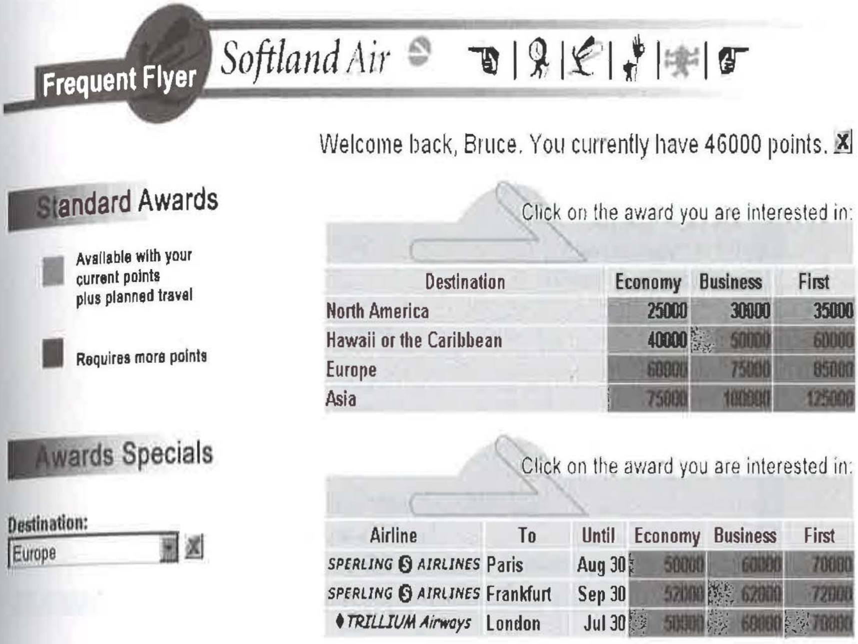
Figure 5-3 "Members, sign in here"

most nearly "fit" with the number of award points you hold. As you do not currently qualify for a trip to Europe, you can use the "Planned Trips" portion of the screen to determine what trips you can take by this summer in order to qualify for the award you want.

Using the screen in Figure 5-4, you can plan trips and even book tickets. In this way you can put enough miles in the bank to be able to earn the award to Europe.

Notice how the entire transaction is personalized for the client interacting with the Web site. It is also important to note that aside from logging on to enter your membership number and select the frequent-flyer transaction, all other transactions occurred on the PC in your home or office. Because the middle-tier server can aggregate data from remote sources, package it as XML documents, and send it to the client, a continuous connection to the servers was not required.

This is quite a different model from what we have today on the Web. And XML, working with programs like SoftQuad's HoTMetaL Application Server in the middle tier, is the reason.



Welcome back, Bruce. You currently have 46000 points. ☒

Standard Awards

▨ Available with your current points plus planned travel

■ Requires more points

Click on the award you are interested in:

Destination	Economy	Business	First
North America	25000	30000	35000
Hawaii or the Caribbean	40000	50000	60000
Europe	60000	75000	85000
Asia	75000	100000	125000

Awards Specials

Destination: Europe ☒

Click on the award you are interested in:

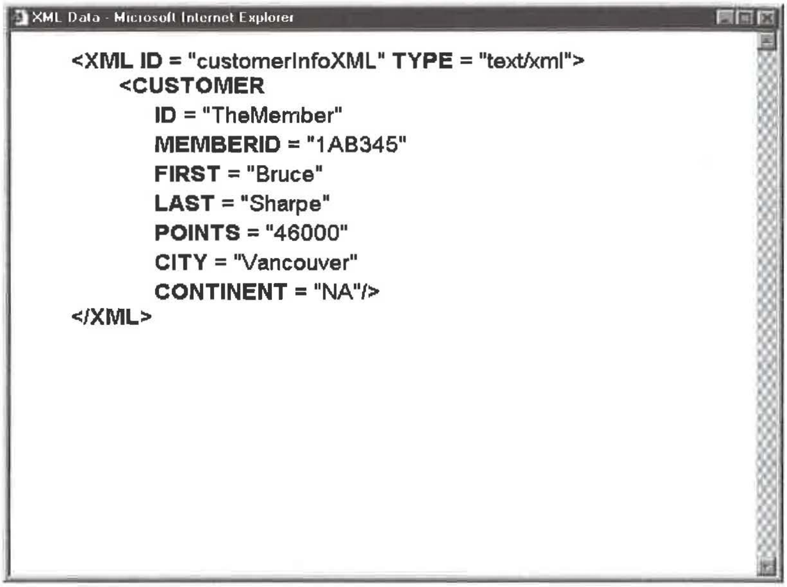
Airline	To	Until	Economy	Business	First
SPERLING AIRLINES	Paris	Aug 30	50000	60000	70000
SPERLING AIRLINES	Frankfurt	Sep 30	52000	62000	72000
TRILLIUM Airways	London	Jul 30	50000	60000	70000

Figure 5-4 Personalized frequent-flyer information

5.5 | Understanding the *Softland Air* scenario

When you connect to the *Softland Air* Web site you first sign in with your membership ID. Your membership number is used to extract your name, the number of award points you have earned, and your point of origin from the "member information" database. This information is sent from remote databases to the middle tier server, which combines it into an XML document (see Figure 5-5). Once the data is in XML, the member name, point of origin, and number of award points can be addressed and used by middle-tier and client applications.

At this point, the middle-tier software knows who signed on. It can request all relevant awards information from both its own awards database and the remote databases maintained by its partner airlines. Figure 5-6 shows the XML data for award specials items from remote awards databases. Note that because this data is in XML, we can easily see the number



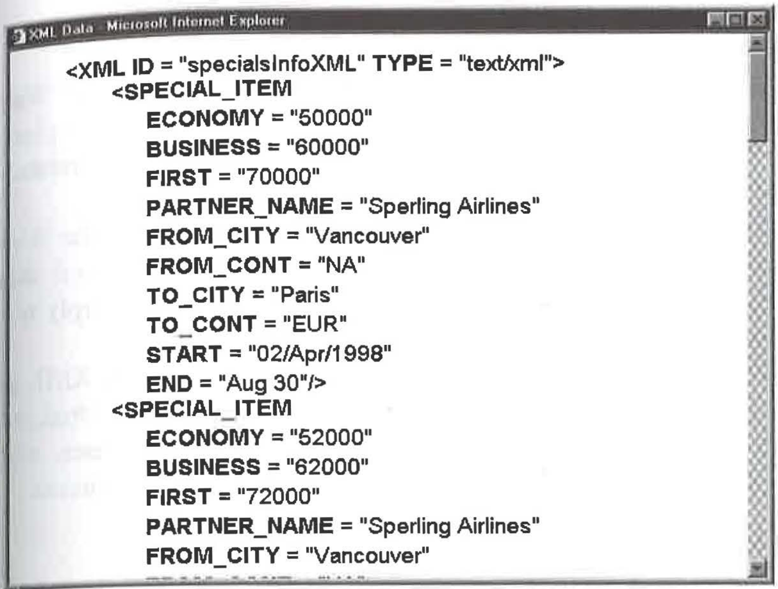
```
<XML ID = "customerInfoXML" TYPE = "text/xml">
    <CUSTOMER
        ID = "TheMember"
        MEMBERID = "1AB345"
        FIRST = "Bruce"
        LAST = "Sharpe"
        POINTS = "46000"
        CITY = "Vancouver"
        CONTINENT = "NA"/>
</XML>
```

Figure 5-5 XML document generated from member information database

of points required for each award, the partner airline name, the point of origin, the destination, and the dates the special runs. Again, this information is available for use by both middle-tier and client-side processing based on member queries.

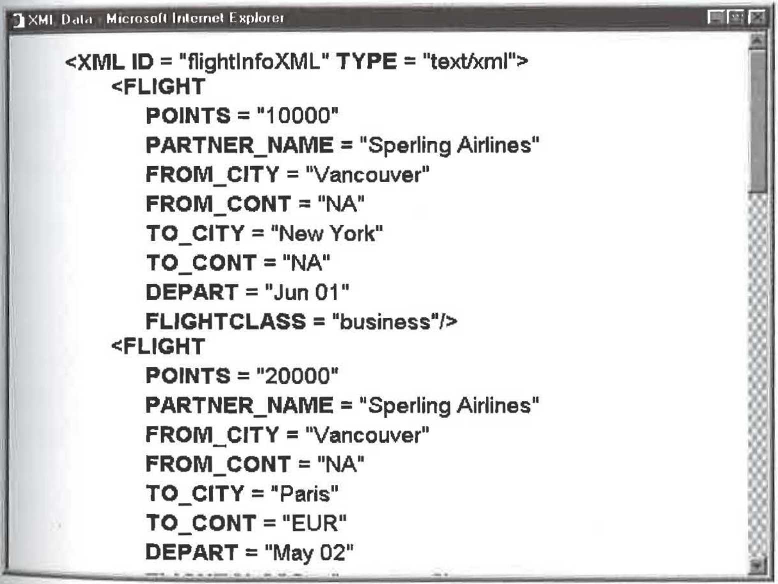
The middle-tier server can also request all flight point earnings information from all remote flight information databases, as shown in Figure 5-7. We can easily see the number of points that would be earned from each flight, the partner airline name, the point of origin, the destination, and the date of flight and class of service. This information is available for use by middle-tier and client-side processing.

The information that is sent to the middle tier is compact, personalized, and precise. It differs from HTML because it contains the actual abstract data, not the look of the screen. Middle-tier software, like the *HoTMetaL Application Server*, acts to assemble and deliver the right information at the right time, minimizing Web traffic and providing a higher degree of user interaction and satisfaction.



```
<XML ID = "specialsInfoXML" TYPE = "text/xml">
    <SPECIAL_ITEM
        ECONOMY = "50000"
        BUSINESS = "60000"
        FIRST = "70000"
        PARTNER_NAME = "Sperling Airlines"
        FROM_CITY = "Vancouver"
        FROM_CONT = "NA"
        TO_CITY = "Paris"
        TO_CONT = "EUR"
        START = "02/Apr/1998"
        END = "Aug 30"/>
    <SPECIAL_ITEM
        ECONOMY = "52000"
        BUSINESS = "62000"
        FIRST = "72000"
        PARTNER_NAME = "Sperling Airlines"
        FROM_CITY = "Vancouver"
```

Figure 5-6 XML document generated from award specials database



```
<XML ID = "flightInfoXML" TYPE = "text/xml">
    <FLIGHT
        POINTS = "10000"
        PARTNER_NAME = "Sperling Airlines"
        FROM_CITY = "Vancouver"
        FROM_CONT = "NA"
        TO_CITY = "New York"
        TO_CONT = "NA"
        DEPART = "Jun 01"
        FLIGHTCLASS = "business"/>
    <FLIGHT
        POINTS = "20000"
        PARTNER_NAME = "Sperling Airlines"
        FROM_CITY = "Vancouver"
        FROM_CONT = "NA"
        TO_CITY = "Paris"
        TO_CONT = "EUR"
        DEPART = "May 02"
```

Figure 5-7 XML document with flight point earnings

5.6 | Towards the Brave New Web

The World Wide Web continues to evolve rapidly. Today the "hottest" Web sites are those that provide multimedia sizzle. But as the shift takes place from simply providing entertainment value to facilitating business transactions, dynamic personalized content will become "hot".

Products like SoftQuad's HoTMetaL Application Server allow the Web site developer to add a new middle tier server to the Web model. It is this middle tier that enables business transactions in a way that was simply not possible before XML.

The *Softland Air* scenario shows how a middle-tier server, using XML as a structured information interchange representation, enables personalized data aggregation and organization from multiple remote databases, and interactive delivery to client browsers based upon end-user requirements.