



---

# XML

## IN A NUTSHELL

*A Desktop Quick Reference*

**O'REILLY®**

*Elliott Rusty Harold & W. Scott Means*  
001 ServiceNow, Inc.'s Exhibit 1010

## ***XML in a Nutshell***

by Elliotte Rusty Harold and W. Scott Means

Copyright © 2001 O'Reilly & Associates, Inc. All rights reserved.  
Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

***Contributor:*** Stephen Spainhour

***Editors:*** Laurie Petrycki and John Posner

***Production Editor:*** Ann Schirmer

***Cover Designer:*** Ellie Volckhausen

***Printing History:***

January 2001: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly & Associates, Inc. The association of the image of a peafowl and the topic of XML is a trademark of O'Reilly & Associates, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

*Library of Congress Cataloging-in-Publication Data can be found at:*  
*<http://www.oreilly.com/catalog/xmlnut>.*

## *Table of Contents*

<i>Preface</i> .....	<i>xi</i>
----------------------	-----------

### *Part I: XML Concepts*

---

<i>Chapter 1—Introducing XML</i> .....	<i>3</i>
What XML Offers .....	3
Portable Data .....	6
How XML Works .....	6
The Evolution of XML .....	8
 <i>Chapter 2—XML Fundamentals</i> .....	 <i>11</i>
XML Documents and XML Files .....	11
Elements, Tags, and Character Data .....	12
Attributes .....	15
XML Names .....	17
Entity References .....	18
CDATA Sections .....	19
Comments .....	20
Processing Instructions .....	20
The XML Declaration .....	21
Checking Documents for Well-Formedness .....	23



<i>Chapter 3—Document Type Definitions</i> .....	26
Validation .....	26
Element Declarations .....	34
Attribute Declarations .....	39
General Entity Declarations .....	46
External Parsed General Entities .....	48
External Unparsed Entities and Notations .....	49
Parameter Entities .....	51
Conditional Inclusion .....	53
Two DTD Examples .....	54
Locating Standard DTDs .....	56
 <i>Chapter 4—Namespaces</i> .....	 58
The Need for Namespaces .....	58
Namespace Syntax .....	61
How Parsers Handle Namespaces .....	66
Namespaces and DTDs .....	67
 <i>Chapter 5—Internationalization</i> .....	 69
The Encoding Declaration .....	69
Text Declarations .....	70
XML-Defined Character Sets .....	71
Unicode .....	72
ISO Character Sets .....	74
Platform-Dependent Character Sets .....	75
Converting Between Character Sets .....	76
The Default Character Set for XML Documents .....	77
Character References .....	78
xml:lang .....	81
 <i>Part II: Narrative-Centric Documents</i>	
 <i>Chapter 6—XML as a Document Format</i> .....	 85
SGML's Legacy .....	85
Narrative Document Structures .....	86
TEI .....	88
DocBook .....	91
Document Permanence .....	94
Transformation and Presentation .....	96

<b>Chapter 7—XML on the Web</b>	<b>98</b>
XHTML	99
Direct Display of XML in Browsers	105
Authoring Compound Documents with Modular XHTML	110
Prospects for Improved Web Search Methods	124
<b>Chapter 8—XSL Transformations</b>	<b>129</b>
An Example Input Document	129
xsl:stylesheet and xsl:transform	130
Stylesheet Processors	132
Templates	133
Calculating the Value of an Element with xsl:value-of	134
Applying Templates with xsl:apply-templates	135
The Built-in Template Rules	138
Modes	142
Attribute Value Templates	144
XSLT and Namespaces	144
Other XSLT Elements	146
<b>Chapter 9—XPath</b>	<b>147</b>
The Tree Structure of an XML Document	147
Location Paths	150
Compound Location Paths	155
Predicates	157
Unabbreviated Location Paths	158
General XPath Expressions	160
XPath Functions	163
<b>Chapter 10—XLinks</b>	<b>168</b>
Simple Links	169
Link Behavior	170
Link Semantics	173
Extended Links	173
Linkbases	180
DTDs for XLinks	181
<b>Chapter 11—XPointers</b>	<b>182</b>
XPointers on URLs	182
XPointers in Links	184
Bare Names	185

Child Sequences .....	186
Points .....	186
Ranges .....	189
 <i>Chapter 12—Cascading Stylesheets (CSS)</i> .....	 191
The Three Levels of CSS .....	193
CSS Syntax .....	193
Associating Stylesheets with XML Documents .....	195
Selectors .....	197
The Display Property .....	200
Pixels, Points, Picas, and Other Units of Length .....	201
Font Properties .....	202
Text Properties .....	203
Colors .....	204
 <i>Chapter 13—XSL Formatting Objects (XSL-FO)</i> .....	 206
XSL Formatting Objects .....	208
The Structure of an XSL-FO Document .....	209
Master Pages .....	210
XSL-FO Properties .....	216
Choosing Between CSS and XSL-FO .....	221
 <i>Part III: Data-Centric Documents</i>	
 <i>Chapter 14—XML as a Data Format</i> .....	 225
Programming Applications of XML .....	225
Describing Data .....	227
Support for Programmers .....	229
 <i>Chapter 15—Programming Models</i> .....	 230
Event- Versus Object-Driven Models .....	230
Programming Language Support .....	231
Non-Standard Extensions .....	232
Transformations .....	232
Processing Instructions .....	233
Links and References .....	233
Notations .....	234
What You Get Is Not What You Saw .....	234



<i>Chapter 16—Document Object Model (DOM)</i> .....	236
DOM Core .....	237
DOM Strengths and Weaknesses .....	237
Parsing a Document with DOM .....	238
The Node Interface .....	238
Specific Node Types .....	240
The DOMImplementation Interface .....	245
A Simple DOM Application .....	245
 <i>Chapter 17—SAX</i> .....	 250
The ContentHandler Interface .....	252
SAX Features and Properties .....	259
 <i>Part IV: Reference</i>	
 <i>Chapter 18—XML 1.0 Reference</i> .....	 265
How to Use This Reference .....	265
Annotated Sample Documents .....	265
Key to XML Syntax .....	266
Well-Formedness .....	266
Validity .....	273
Global Syntax Structures .....	279
DTD (Document Type Definition) .....	285
Document Body .....	294
XML Document Grammar .....	295
 <i>Chapter 19—XPath Reference</i> .....	 299
The XPath Data Model .....	299
Datatype .....	300
Location Paths .....	301
Predicates .....	305
XPath Functions .....	305
 <i>Chapter 20—XSLT Reference</i> .....	 315
The XSLT Namespace .....	315
XSLT Elements .....	315
XSLT Functions .....	339

<i>Chapter 21—DOM Reference</i> .....	345
Object Hierarchy .....	346
Object Reference .....	346
<i>Chapter 22—SAX Reference</i> .....	400
The org.xml.sax Package .....	400
The org.xml.sax.helpers Package .....	407
SAX Features and Properties .....	413
The org.xml.sax.ext Package .....	415
<i>Chapter 23—Character Sets</i> .....	417
Character Tables .....	419
HTML4 Entity Sets .....	424
Other Unicode Blocks .....	432
<i>Index</i> .....	459





## *Preface*

XML is one of the most important developments in document syntax in the history of computing. In the last few years it has been adopted in fields as diverse as law, aeronautics, finance, insurance, robotics, multimedia, hospitality, travel, art, construction, telecommunications, software design, agriculture, physics, journalism, theology, retail, and medieval literature. XML has become the syntax of choice for newly designed document formats across almost all computer applications. It's used on Linux, Windows, Macintosh, and many other computer platforms. Mainframes on Wall Street trade stocks with one another by exchanging XML documents. Children playing games on their home PCs save their documents in XML. Sports fans receive real-time game scores on their cell phones in XML. XML is simply the most robust, reliable, and flexible document syntax ever invented.

*XML in a Nutshell* is a comprehensive guide to the rapidly growing world of XML. It covers all aspects of XML, from the most basic syntax rules, to the details of DTD creation, to the APIs you can use to read and write XML documents in a variety of programming languages.

### *What This Book Covers*

There are hundreds of formally established XML applications from the W3C and other standards bodies, such as OASIS and the Object Management Group. There are even more informal, unstandardized applications from individuals and corporations, such as Microsoft's Channel Definition Format and John Guajardo's Mind Reading Markup Language. This book cannot cover them all, any more than a book on Java could discuss every program that has ever been or might ever be written in Java. This book focuses primarily on XML itself. It covers the fundamental rules that all XML documents and authors must adhere to, whether a web designer uses SMIL to add animations to web pages or a C++ programmer uses SOAP to serialize objects into a remote database.

This book also covers generic supporting technologies that have been layered on top of XML and are used across a wide range of XML applications. These technologies include:

#### *XLinks*

An attribute-based syntax for hyperlinks between XML and non-XML documents that provide the simple, one-directional links familiar from HTML, multidirectional links between many documents, and links between documents you don't have write access to.

#### *XSLT*

An XML application that describes transformations from one document to another, in either the same or different XML vocabularies.

#### *XPointers*

A syntax for identifying particular parts of an XML document referred to by a URI; often used in conjunction with an XLink.

#### *XPath*

A non-XML syntax used by both XPointers and XSLT for identifying particular pieces of XML documents. For example, an XPath can locate the third `address` element in the document, or all elements with an `email` attribute whose value is `elharo@metalab.unc.edu`.

#### *Namespaces*

A means of distinguishing between elements and attributes from different XML vocabularies that have the same name; for instance, the title of a book and the title of a web page in a web page about books.

#### *SAX*

The Simple API for XML, an event-based Java application programming interface implemented by many XML parsers.

#### *DOM*

The Document Object Model, a tree-oriented API that treats an XML document as a set of nested objects with various properties.

All these technologies, whether defined in XML (XLinks, XSLT, and Namespaces) or in another syntax (XPointers, XPath, SAX, and DOM), are used in many different XML applications.

This book does not specifically cover XML applications that are relevant to only some users of XML. These include:

#### *SVG*

Scalable Vector Graphics is a W3C-endorsed standard used for encoding line drawings in XML.

#### *MathML*

The Mathematical Markup Language is a W3C-endorsed standard XML application used for embedding equations in web pages and other documents.



### *CML*

The Chemical Markup Language was one of the first XML applications. It describes chemistry, solid-state physics, molecular biology, and the other molecular sciences.

### *RDF*

The Resource Description Framework is a W3C-standard XML application used for describing resources, with a particular focus on the sort of metadata one might find in a library card catalog.

### *CDF*

The Channel Definition Framework is a nonstandard, Microsoft-defined XML application used to publish web sites to Internet Explorer for offline browsing.

Occasionally we use one or more of these applications in an example, but we do not cover all aspects of the relevant vocabulary in depth. While interesting and important, these applications (and hundreds more like them) are intended primarily for use with special software that knows their format intimately. For instance, graphic designers do not work directly with SVG. Instead, they use their customary tools, such as Adobe Illustrator, to create SVG documents. They may not even know they're using XML.

This book focuses on standards that are relevant to almost all developers working with XML. We investigate XML technologies that span a wide range of XML applications, not those that are relevant only within a few restricted domains.

## *Organization of the Book*

Part I, *XML Concepts*, introduces you to the fundamental standards that form the essential core that all XML applications and software must adhere to. It teaches you about well-formed XML, DTDs, namespaces, and Unicode as quickly as possible.

Part II, *Narrative-Centric Documents*, explores technologies that are used mostly for narrative XML documents, such as web pages, books, articles, diaries, and plays. You'll learn about XSLT, CSS, XSL-FO, XLinks, XPointers, and XPath.

One of the most unexpected developments in XML was its enthusiastic adoption of data-heavy structured documents such as spreadsheets, financial statistics, mathematical tables, and software file formats. Part III, *Data-Centric XML*, explores the use of XML for such data-intensive documents. This part focuses on the tools and APIs needed to write software that process XML, including SAX, the Simple API for XML, and the W3C's Document Object Model.

Finally, Part IV, *Reference*, is a series of quick-reference chapters that form the core of any Nutshell handbook. These chapters give you detailed syntax rules for the core XML technologies, including XML, DTDs, XPath, XSLT, SAX, and DOM. Turn to this section when you need to quickly find out the precise syntax for something you know you can do but don't remember exactly how to do.



## Conventions Used in This Book

Body text, like the text you're reading now, is written in Garamond.

Constant width is used for:

- Code examples and fragments.
- Anything that might appear in an XML document, including element names, tags, attribute values, entity references, and processing instructions.
- Anything that might appear in a program, including keywords, operators, method names, class names, and literals.

### **Constant-width bold**

- User input.
- Signifies emphasis should be deleted.

*Constant-width italic* is used for:

- Replaceable elements in code statements.

*Italic* is used for:

- New terms where they are defined.
- Pathnames, filenames, and program names. (However, if the program name is also the name of a Java class, it is written in constant-width font, like other class names.)
- Host and domain names (*www.xml.com*).
- URLs (*http://ibiblio.org/xml/*).

Significant code fragments, complete programs, and documents are generally placed into a separate paragraph like this:

```
<?xml version="1.0"?>
<?xml-stylesheet href="person.css" type="text/css"?>
<person>
  Alan Turing
</person>
```

XML is case sensitive. The `PERSON` element is not the same thing as the `person` or `Person` element. Case-sensitive languages do not always allow authors to adhere to standard English grammar. It is usually possible to rewrite the sentence so the two do not conflict, and when possible we have endeavored to do so. However, on rare occasions when there is simply no way around the problem, we let standard English come up the loser.

Finally, although most of the examples used here are toy examples unlikely to be reused, a few have real value. Please feel free to reuse them or any parts of them in your own code. No special permission is required. As far as we are concerned, they are in the public domain (though the same is definitely not true of the explanatory text).

## *Request for Comments*

We enjoy hearing from readers with general comments about how this book could be better, specific corrections, or topics you would like to see covered. You can reach the authors by sending email to [elbaro@metalab.unc.edu](mailto:elbaro@metalab.unc.edu) and [smeans@enterprisewebmachines.com](mailto:smeans@enterprisewebmachines.com). Please realize, however, that we each receive several hundred pieces of email a day and cannot respond to every one personally. For the best chances of getting a personal response, please identify yourself as a reader of this book. And please send the message from the account you want us to reply to and make sure that your Reply-to address is properly set. There's nothing quite so frustrating as spending an hour or more carefully researching the answer to an interesting question and composing a detailed response, only to have it bounce because the correspondent sent the message from a public terminal and neglected to set the browser preferences to include their actual email address.

The information in this book has been tested and verified, but you may find that features have changed (or you may even find mistakes). We believe the old saying, "If you like this book, tell your friends. If you don't like it, tell us." We're especially interested in hearing about mistakes. As hard as the authors and editors worked on this book, inevitably there are a few mistakes and typographical errors that slipped by us. If you find a mistake or a typo, please let us know so we can correct it. You can send any errors you find, as well as suggestions for future editions, to:

O'Reilly & Associates, Inc.  
101 Morris Street  
Sebastopol, CA 95472  
1-800-998-9938 (in the United States or Canada)  
1-707-829-0515 (international/local)  
1-707-829-0104 (fax)

We have a web site for the book, where we list errata, examples, and any additional information. You can access this site at:

<http://www.oreilly.com/catalog/xmlnut>

Before reporting errors, please check this web site to see if we already posted a fix. To ask technical questions or comment on the book, send email to:

[bookquestions@oreilly.com](mailto:bookquestions@oreilly.com)

For more information about our books, conferences, software, Resource Centers, and the O'Reilly Network, see our web site at:

<http://www.oreilly.com>

## *Acknowledgments*

Many people were involved in the production of this book. The original editor, John Posner, got this book rolling and provided many helpful comments that substantially improved the book. When John moved on, Laurie Petrycki shepherded this book to its completion. Stephen Spainhour deserves special thanks for



his work on the reference section. His efforts in organizing and reviewing material helped create a better book. We'd like to thank Matt Sergeant and Didier P. H. Martin for their thorough technical review of the manuscript and thoughtful suggestions.

We'd also like to thank everyone who has worked so hard to make XML such a success over the last few years and thereby give us something to write about. There are so many of these people that we can only list a few. In alphabetical order we'd like to thank Tim Berners-Lee, Jon Bosak, Tim Bray, James Clark, Charles Goldfarb, Jason Hunter, Michael Kay, Brett McLaughlin, David Megginson, David Orchard, Walter E. Perry, Simon St. Laurent, C. M. Sperberg-McQueen, James Tauber, B. Tommie Usdin, and Mark Wutka. Our apologies to everyone we unintentionally omitted.

Elliote would like to thank his agent, David Rogelberg, who convinced him that it was possible to make a living writing books like this rather than working in an office. The entire Sunsite crew (now *ibiblio.org*) has also helped him to communicate better with his readers in a variety of ways over the last several years. All these people deserve much thanks and credit. Finally, as always, he offers his largest thanks to his wife, Beth, without whose love and support this book would never have happened.

Scott would most like to thank his lovely wife, Celia, who has already spent way too much time as a "computer widow." He would also like to thank his daughter Selene for understanding why Daddy can't play with her when he's "working," and Skyler for just being himself. Also, he'd like to thank the team at Enterprise Web Machines for helping him make time to write. Finally, he would like to thank John Posner for getting him into this and Laurie Petrycki for working with him when things got tough.

Elliote Rusty Harold  
*elharo@metalab.unc.edu*

W. Scott Means  
*smeans@enterprisewebmachines.com*





## CHAPTER 1

# *Introducing XML*

XML, the Extensible Markup Language, is a W3C-endorsed standard for document markup. It defines a generic syntax used to mark up data with simple, human-readable tags. It provides a standard format for computer documents. This format is flexible enough to be customized for domains as diverse as web sites, electronic data interchange, vector graphics, genealogy, real estate listings, object serialization, remote procedure calls, and voice mail systems.

You can write your own programs that interact with, massage, and manipulate data in XML documents. If you do, you'll have access to a wide range of free libraries in a variety of languages that can read and write XML so that you can focus on the unique needs of your program. Or you can use off-the-shelf software like web browsers and text editors to work with XML documents. Some tools are able to work with any XML document. Others are customized to support a particular XML application in a particular domain like vector graphics and may not be of much use outside that domain. But in all cases, the same underlying syntax is used, even if it's deliberately hidden by more user-friendly tools or restricted to a single application.

### *What XML Offers*

XML is a meta-markup language for text documents. Data is included in XML documents as strings of text, and the data is surrounded by text markup that describes the data. A particular unit of data and markup is called an *element*. The XML specification defines the exact syntax this markup must follow: how elements are delimited by tags, what a tag looks like, what names are acceptable for elements, where attributes are placed, and so forth. Superficially, the markup in an XML document looks much like that in an HTML document, but some crucial differences exist.

Most importantly, XML is a *meta-markup language*. That means it doesn't have a fixed set of tags and elements that are always supposed to work for everyone in

all areas of interest. Attempts to create a finite set of such tags are doomed to failure. Instead, XML allows developers and writers to define the elements they need as they need them. Chemists can use tags that describe elements, atoms, bonds, reactions, and other items encountered in chemistry. Real estate agents can use elements that describe apartments, rents, commissions, locations, and other items needed in real estate. Musicians can use elements that describe quarter notes, half notes, G clefs, lyrics, and other objects common in music. The *X* in XML stands for *Extensible*. Extensible means that the language can be extended and adapted to meet many different needs.

Although XML is flexible in the elements it allows to be defined, it is strict in many other respects. It provides a grammar for XML documents that regulates placement of tags, where tags appear, which element names are legal, how attributes are attached to elements, and so forth. This grammar is specific enough to allow development of XML parsers that can read and understand any XML document. Documents that satisfy this grammar are said to be *well-formed*. Documents that are not well-formed are not allowed any more than a C program containing a syntax error would be. XML processors reject documents that contain well-formedness errors.

To enhance interoperability, individuals or organizations may agree to use only certain tags. These tag sets are called *XML applications*. An XML application is not a software application that uses XML, like Mozilla or Microsoft Word. Rather, it's an application of XML to a particular domain, such as vector graphics or cooking.

The markup in an XML document describes the document's structure. It lets you see which elements are associated with which other elements. In a well-designed XML document, the markup also describes the document's semantics. For instance, the markup can indicate that an element is a date, a person, or a bar code. In well-designed XML applications, the markup says nothing about how the document should be displayed. That is, it does not say that an element is bold, italicized, or a list item. XML is a structural and semantic markup language, not a presentation language.\*

The markup permitted in a particular XML application can be documented in a document type definition (DTD). The DTD lists all legal markup and specifies where and how the markup may be included in a document. Particular document instances can be compared to the DTD. Documents that match the DTD are said to be *valid*. Documents that do not match are *invalid*. Validity depends on the DTD; whether a document is valid or invalid depends on which DTD you compare it to.

Not all documents need to be valid. For many purposes, a well-formed document is enough. DTDs are optional in XML. On the other hand, DTDs may not always be sufficient. The DTD syntax is limited and does not allow you to make many

---

\* A few XML applications, like XSL Formatting Objects, are designed to describe text presentation. However, these are exceptions that prove the rule. Although XSL-FO describes presentation, you'd never write an XSL-FO document directly. Instead, you'd write a more semantically marked-up XML document, then use an XSL Transformations stylesheet to change the semantic-oriented XML into presentation-oriented XML.



useful statements such as, "This element contains a number" or "This string of text is a date between 1974 and 2032." If you're writing programs to read XML documents, you may want to add code to verify statements like these, just as you would if you were writing code to read a tab-delimited text file. The difference is that XML parsers present you with the data in a much more convenient format to work with and do more of the work for you before you have to resort to your own custom code.

## *What XML Is Not*

XML is a markup language, and only a markup language. It's important to remember this fact. The XML hype has become so extreme that some people expect XML to do everything up to, and including, washing the family dog.

First of all, XML is not a programming language. There's no such thing as an XML compiler that reads XML files and produces executable code. You might define a scripting language that uses a native XML format and is interpreted by a binary program, but even this application would be unusual.\* XML can be used as an instruction format for programs that make things happen. A traditional program, for example, may read a text config file and take different actions, depending on what it sees in the file. There's no reason why a config file can't be written in XML instead of unstructured text. Indeed, some recent programs are beginning to use XML config files. But in all cases the program, not the XML document, takes action. An XML document simply *is*. It does not *do* anything.

Furthermore, XML is not a network-transport protocol. XML, like HTML, won't send data across the network. However, data sent across the network using HTTP, FTP, NFS, or some other protocol might be in an XML format. XML can be the format for data sent across the network, but again, software outside the XML document must actually do the sending.

Finally, to mention the example in which the hype most often obscures reality, XML is not a database. You won't replace an Oracle or MySQL server with XML. A database can contain XML data as a VARCHAR, a BLOB, or a custom XML datatype; but the database itself is not an XML document. You can store XML data in a database or on a server or retrieve data from a database in an XML format, but to do so you need to run software written in a real programming language like C or Java. To store XML in a database, software on the client side sends the XML data to the server using an established network protocol like TCP/IP. Software on the server side receives the XML data, parses it, and stores it in the database. To retrieve an XML document from a database, you generally pass through a middleware product like Enhydra that makes SQL queries against the database and formats the result set as XML before returning it to the client. Indeed, some databases may integrate this software code into their core server or provide plug-ins, such as the Oracle XSQL servlet, to do it. XML serves very well as a ubiquitous, platform-independent transport format in these scenarios. However, XML is not the database and shouldn't be used as one.

---

\* At least one XML application, XSL Transformations, has been proven to be Turing complete.



## *Portable Data*

XML offers the tantalizing possibility of truly cross-platform, long-term data formats. It's long been the case that a document written by one piece of software on one platform is not necessarily readable on a different platform, by a different program on the same platform, or even by a future or past version of the same software on the same platform. When the document can be read, all the information may not necessarily come across. Much of the data from the original moon landings in the late 1960s and early 1970s is now effectively lost. Even if you can find a tape drive that reads the now obsolete tapes, nobody knows what format the data on the tapes is stored in!

XML is an incredibly simple, well-documented, straightforward data format. XML documents are text, and any tool that can read a text file can read an XML document. Both XML data and markup are text, and the markup is present in the XML file as tags. You don't have to wonder whether every eighth byte is random padding, guess whether a four-byte quantity is a two's complement integer or an IEEE 754 floating point number, or try to decipher which integer codes map to which formatting properties. You can read the tag names directly to see exactly what's in the document. Similarly, since tags define element boundaries, you aren't likely to get tripped up by unexpected line ending conventions or the number of spaces mapped to a tab. All the important details about the document's structure are explicit. You don't have to reverse engineer the format or rely on questionable, and often unavailable, documentation.

A few software vendors may want to lock in their users with undocumented, proprietary binary file formats. However, in the long run we are all better off if we can use the cleanly documented, well-understood, easy to parse, text-based formats that XML provides. XML allows documents and data to move from one system to another with a reasonable hope that the receiving system can make sense out of it. Furthermore, validation lets the receiving side ensure that it gets what it expects. Java promised portable code. XML delivers portable data. In many ways, XML is the most portable and flexible document format designed since the ASCII text file.

## *How XML Works*

Example 1-1 shows a simple XML document. This particular XML document might appear in an inventory control system or a stock database. It marks up the data with tags and attributes describing the color, size, bar code number, manufacturer, and product name.

### *Example 1-1: An XML Document*

```
<?xml version="1.0"?>
<product barcode="2394287410">
  <manufacturer>Verbatim</manufacturer>
  <name>DataLife MF 2HD</name>
  <quantity>10</quantity>
  <size>3.5"</size>
```

*Example 1-1: An XML Document (continued)*

```
<color>black</color>
<description>floppy disks</description>
</product>
```

This document is text and might well be stored in a text file. You can edit this file with any standard text editor, such as BBEdit, UltraEdit, Emacs, or vi. You do not need a special XML editor; in fact, we find that most general-purpose XML editors are far more trouble than they're worth and much harder to use than a simple text editor.

Then again, this document might not be a file at all. It might be a record in a database. It might be assembled on the fly by a CGI query to a web server and exist only in a computer's memory. It might even be stored in multiple files and assembled at runtime. Even if it isn't in a file, however, the document is a text document that can be read and transmitted by any software capable of reading and transmitting text.

Programs that actually try to understand the contents of the XML document, that is, do not merely treat it as any other text file, use an *XML parser* to read the document. The parser is responsible for dividing the document into individual elements, attributes, and other pieces. It passes the contents of the XML document to the application piece by piece. If at any point the parser detects a violation of XML rules, it reports the error to the application and stops parsing. In some cases the parser may read past the original error in the document so it can detect and report other errors that occur later in the document. However, once it has detected the first error, it no longer passes along the contents of the elements and attributes it encounters to the application.

Individual XML applications normally dictate precise rules about which elements and attributes are allowed where. You wouldn't expect to find a `G_Clef` element when reading a biology document, for instance. Some of these rules can be specified precisely using a DTD. A document may contain either the DTD itself or a pointer to a URI where the DTD is found. Some XML parsers notice these details and compare the document to its DTD as they read it to see if the document satisfies the specified constraints. Such a parser is called a *validating parser*. A violation of those constraints is a *validity error*, and the whole process of checking a document against a DTD is called *validation*. If a validating parser finds a validity error, it reports it to the application on whose behalf it parses the document. This application can then decide whether it wishes to continue parsing the document. However, validity errors, unlike well-formedness errors, are not necessarily fatal; an application may choose to ignore them. Not all parsers are validating parsers. Some merely check for well-formedness.

The application that receives data from the parser may be:

- A web browser, such as Netscape or Internet Explorer, that displays the document to a reader
- A word processor, such as StarOffice Writer, that loads the XML document for editing
- A database server, such as Oracle, that stores XML data in a database



- A drawing program, such as Corel Draw, that interprets XML as two-dimensional coordinates for the contents of a picture
- A spreadsheet, such as Gnumeric, that parses XML to find numbers and functions used in a calculation
- A personal finance program, such as Microsoft Money, that sees XML as a bank statement
- A syndication program that reads the XML document and extracts the headlines for today's news
- A program that you wrote in Java, C, Python, or some other language that does exactly what you want it to do
- Almost anything else

XML is an *extremely* flexible format for data. It can be used in all of these scenarios and many more. These examples are real. In theory, any data that can be stored in a computer can be stored in XML format. In practice, XML is suitable for storing and exchanging any data that can be plausibly encoded as text. Its use is unsuitable only for multimedia data, such as photographs, recorded sound, video, and other very large bit sequences.

## *The Evolution of XML*

XML is a descendant of the Standard Generalized Markup Language (SGML). The language that would eventually become SGML was invented by Charles Goldfarb, Ed Mosher, and Ray Lorie at IBM in the 1970s and developed by several hundred people around the world until its eventual adoption as ISO standard 8879 in 1986. SGML was intended to solve many of the same problems XML solves. It was and is a semantic and structural markup language for text documents. SGML is extremely powerful and achieved some success in the U.S. military and government, the aerospace sector, and other domains that needed ways of managing technical documents that were tens of thousands of pages long efficiently.

SGML's biggest success was HTML, an SGML application. However, HTML is just one SGML application. It does not have or offer the full power of SGML. Since HTML restricts authors to a finite set of tags designed to describe web pages in a fairly presentationally oriented way, it's really little more than a traditional markup language that has been adopted by web browsers. It simply doesn't lend itself to use beyond the single application of web page design. You would not use HTML to exchange data between incompatible databases or send updated product catalogs to retailer sites, for example. HTML is useful for creating web pages, but it isn't capable of much more than that.

The obvious choice for other applications that took advantage of the Internet, but were not simple web pages, was SGML. SGML's main problem is its complexity. The official SGML specification is more than 150 very technical pages. It covers many special cases and unlikely scenarios. It is so complex that almost no software has ever implemented it fully. Programs that implemented or relied on different subsets of SGML were often incompatible with one another. The special feature one program considered essential would be considered extraneous fluff and omitted by the next program.



In 1996 Jon Bosak, Tim Bray, C. M. Sperberg-McQueen, James Clark, and several others began work on a "lite" version of SGML. This version retained most of SGML's power, but trimmed many features that were redundant, too complicated to implement, confusing to end users, or that had simply not been proven useful over the previous 20 years of experience with SGML. The result, in February 1998, was XML 1.0, and it was an immediate success. Many developers who knew they needed a structural markup language but couldn't bring themselves to accept SGML's complexity adopted XML wholeheartedly. It was ultimately used in domains ranging from legal court filings to hog farming.

However, XML 1.0 was just the beginning. The next standard out of the gate was Namespaces in XML, an effort to allow conflict-free use of markup from different XML applications in the same document. A web page about books, for example, could have a `title` element that referred to the page's title and `title` elements that referred to the book's title, and the two would not conflict.

The Extensible Stylesheet Language, an XML application that transforms other XML documents into a form that is viewable in web browsers, was the next development. This language soon split into XSL Transformations (XSLT) and XSL Formatting Objects (XSL-FO). XSLT has become a general-purpose language for transforming one XML document into another for web page display and other purposes. XSL-FO is an XML application that describes the layout of both printed and web pages. This application rivals PostScript for its power and expressiveness.

However, XSL is not the only option for styling XML documents. The Cascading Stylesheet Language (CSS) was already in use for HTML documents when XML was invented, and it was a reasonable fit to XML, as well. With the advent of CSS Level 2, the W3C made styling XML documents an explicit goal for CSS and gave it equal importance to HTML. The preexisting Document Style Sheet and Semantics Language (DSSSL) was also adopted from its roots in the SGML world to style XML documents for print and use on the Web.

The Extensible Linking Language (XLL) defined more powerful linking constructs that could connect XML documents in a hypertext network, vastly overpowering HTML's `A` tag. It also divided into two separate standards: XLink, which described connections between documents, and XPointer, which addressed the individual parts of an XML document. At this point, it was noticed that both XPointer and XSLT were developing fairly sophisticated, yet incompatible, syntaxes to do exactly the same thing: identify particular elements of an XML document. Consequently, the addressing parts of both specifications were split off and combined into a third specification, XPath.

A similar phenomenon occurred when it was noticed that XML 1.0, XSLT, XML Schemas, and the Document Object Model (DOM) all had similar, but subtly different, conceptual models of the structure of an XML document. For instance, XML 1.0 considers a document's root element as its root, while XSLT uses a more abstract root that includes the root element and several other pieces. Thus the W3C XML Core Working Group began work on an XML Information Set that all these standards could rely on and refer to.

Another piece of the puzzle was a uniform interface for accessing the contents of the XML document from inside a Java, JavaScript, or C++ program. The simplest

API was to merely treat the document as an object that contained other objects. Indeed, work was already underway inside and outside the W3C to define such a Document Object Model for HTML. Expanding this effort to cover XML was not difficult.

Outside the W3C, Peter Murray-Rust, David Megginson, Tim Bray, and other members of the *xml-dev* mailing list recognized that XML parsers, while all compatible in the documents they could parse, were incompatible in their APIs. This observation led to the development of the Simple API for XML, SAX. SAX2 was released in 2000 to add greater configurability, namespace support, and several optional features.

Development of extensions to the core XML specification continues. Future directions include:

*XFragment*

An effort to make sense out of XML document pieces that may not be considered well-formed documents in isolation.

*XML Schemas*

An XML application that can describe the allowed content of documents conforming to a particular XML vocabulary.

*XHTML*

A reformulation of HTML as a well-formed, modular, potentially valid XML application.

*XML Query Language*

A language for finding the elements in a document that meet specified criteria.

*Canonical XML*

A standard algorithm used for determining whether two XML documents are the same after throwing away insignificant details, such as whether single or double quotes are used around attribute values.

*XML Signatures*

A standard means of digitally signing XML documents, embedding signatures in XML documents, and authenticating the resulting documents.

Many new extensions of XML remain to be invented. XML has proven itself a solid foundation for many other technologies.