

THIRD EDITION

Steven Holzner

SAMS
Teach Yourself

XML

SAMS

in 21 Days

www.EngineeringBooksPdf.com

Steven Holzner

SAMS

Teach Yourself

XML

in 21 Days

THIRD EDITION

SAMS

800 East 96th Street, Indianapolis, Indiana, 46240 USA

www.EngineeringBooksPdf.com

Sams Teach Yourself XML in 21 Days, Third Edition

Copyright © 2004 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-32576-4

Library of Congress Catalog Card Number: 2003110401

Printed in the United States of America

First Printing: October 2003

06 05 04 03 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
1-317-428-3341
international@pearsontechgroup.com

ASSOCIATE PUBLISHER

Michael Stephens

ACQUISITIONS EDITOR

Todd Green

DEVELOPMENT EDITOR

Songlin Qiu

MANAGING EDITOR

Charlotte Clapp

PROJECT EDITOR

Matthew Purcell

INDEXER

Mandie Frank

PROOFREADER

Paula Lowell

TECHNICAL EDITOR

Chris Kenyeres

TEAM COORDINATOR

Cindy Teeters

INTERIOR DESIGNER

Gary Adair

COVER DESIGNER

Gary Adair

PAGE LAYOUT

Michelle Mitchell

Contents at a Glance

Introduction	1
Part I Creating XML Documents	7
Day 1 Welcome to XML	9
2 Creating XML Documents	43
3 Creating Well-Formed XML Documents	77
4 Creating Valid XML Documents: DTDs	107
5 Handling Attributes and Entities in DTDs	141
6 Creating Valid XML Documents: XML Schemas	179
7 Creating Your Own Types in XML Schemas	207
Part I In Review	239
Part II Formatting XML Documents	247
Day 8 Formatting XML by Using Cascading Style Sheets	249
9 Formatting XML by Using XSLT	285
10 Working with XSL Formatting Objects	331
Part II In Review	371
Part III XML at Work	379
Day 11 Extending HTML with XHTML	381
12 Putting XHTML to Work	419
13 Creating Graphics and Multimedia: SVG and SMIL	449
14 Handling XLinks, Xpointer, and XForms	485
Part III In Review	521
Part IV Programming and XML	529
Day 15 Using JavaScript and XML	531
16 Using Java and .NET: DOM	567

17	Using Java and .NET: SAX	607
18	Working with SOAP and RDF	645
Part IV In Review		683
Part V Data Handling and XML		691
Day 19	Handling XML Data Binding	693
20	Working with XML and Databases	727
21	Handling XML in .NET	765
Part V In Review		801
Appendix A	Answers to Quiz Questions	807
	Index	817

Table of Contents

Introduction	1
PART I At a Glance	7
DAY 1 Welcome to XML	9
All About Markup Languages	10
All About XML	12
Looking at XML in a Browser	17
Working with XML Data Yourself	20
Structuring Your Data	24
Creating Well-Formed XML Documents	24
Creating Valid XML Documents	25
How XML Is Used in the Real World	26
Using XML: Mathematical Markup Language	27
Using XML: Chemical Markup Language	28
Using XML: Synchronized Multimedia Integration Language	29
Using XML: XHTML	30
Using XML: HTML+TIME	32
Using XML: Microsoft's .NET	33
Using XML: Scalable Vector Graphics	35
Using XML: SOAP	36
Online XML Resources	37
Summary	40
Q&A	41
Workshop	41
Quiz	41
DAY 2 Creating XML Documents	43
Choosing an XML Editor	44
Using XML Browsers	46
Using XML in Internet Explorer	47
Using XML in Netscape Navigator	47
Using XML in Jumbo	48
Using XML Validators	48
Creating XML Documents Piece by Piece	51
Character Encodings: ASCII, Unicode, and UCS	52
Understanding XML Markup and XML Data	55
Using Whitespace and Ends of Lines	57
Creating Prologs	58

Creating an XML Declaration	59
Creating XML Comments	60
Creating Processing Instructions	62
Creating Tags and Elements	63
Creating Tag Names	63
Creating Empty Elements	64
Creating a Root Element	64
Creating Attributes	65
Creating CDATA Sections	70
Handling Entities	73
Summary	74
Q&A	74
Workshop	75
Quiz	75
Day 3 Creating Well-Formed XML Documents	77
What Makes an XML Document Well-Formed?	78
Matching the Production Labeled document	79
Meeting the Well-Formedness Constraints	80
Making Parsed Entity Must Be Well-Formed	80
Creating an Example XML Document	80
Understanding the Well-Formedness Constraints	84
Beginning the Document with an XML Declaration	84
Using Only Legal Character References	85
Including at Least One Element	85
Structuring Elements Correctly	85
Using the Root Element to Contain All Other Elements	87
Nesting Elements Properly	87
Making Attribute Names Unique	88
Enclose Attribute Values in Quotation Marks	89
Avoiding Entity References and < in Attribute Values	89
Avoiding Overuse of < and &	90
Using XML Namespaces	92
Creating Namespaces	92
Defining Namespaces with URIs	93
Creating Local Namespaces	97
Creating Default Namespaces	98
Understanding XML Infosets	102
Understanding Canonical XML	103
Summary	104
Q&A	105
Workshop	105
Quiz	105

DAY 4 Creating Valid XML Documents: DTDs	107
All About DTDs	108
Validating a Document by Using a DTD	112
Creating Element Content Models	113
Handling Any Content	114
Specifying Child Elements	114
Handling Text Content	116
Specifying Multiple Child Elements	118
Allowing Mixed Content	124
Allowing Empty Elements	126
Commenting a DTD	127
Supporting External DTDs	128
Private and Public DTDs	128
Using Internal and External DTDs at the Same Time	131
Handling Namespaces in DTDs	133
Summary	136
Q&A	137
Workshop	137
Quiz	137
Exercises	139
 DAY 5 Handling Attributes and Entities in DTDs	 141
Declaring Attributes in DTDs	142
Using the Legal Default Values and Attribute Types	145
Specifying Default Values	146
Immediate Values	146
The #REQUIRED Default Value	147
The #IMPLIED Default Value	148
The #FIXED Default Value	149
Specifying Attribute Types	150
The CDATA Attribute Type	150
Enumerated Types	151
The NMTOKEN Attribute Type	152
The NMTOKENS Attribute Type	153
The ID Attribute Type	154
The IDREF Attribute Type	155
The ENTITY Attribute Type	156
The ENTITIES Attribute Type	157
The NOTATION Attribute Type	158
Handling Entities	160
Creating Internal General Entity References	162
Creating External General Entity References	165

Associating Non-XML Data with an XML Document	168
Creating Internal Parameter Entities	171
Creating External Parameter Entities	172
Using INCLUDE and IGNORE to Parameterize DTDs	174
Summary	176
Q&A	177
Workshop	177
Quiz	177
Exercises	178
Day 6 Creating Valid XML Documents: XML Schemas	179
Using XML Schema Tools	181
Creating Schemas by Using XML Schema-Creation Tools	181
Validating XML Documents by Using XML Schemas	184
Creating XML Schemas	189
Dissecting an XML Schema	192
The Built-in XML Schema Elements	193
Creating Elements and Types	195
Using Simple Types	195
Using Complex Types	198
Specifying a Number of Elements	200
Specifying Element Default Values	201
Creating Attributes	202
Summary	203
Q&A	204
Workshop	205
Quiz	205
Exercises	205
Day 7 Creating Types in XML Schemas	207
Restricting Simple Types by Using XML Schema Facets	208
Creating XML Schema Choices	214
Using Anonymous Type Definitions	215
Declaring Empty Elements	217
Declaring Mixed-Content Elements	218
Grouping Elements Together	219
Grouping Attributes Together	221
Declaring all Groups	222
Handling Namespaces in Schemas	222
Declaring Locals Without Qualifying Them	224
Declaring and Qualifying Locals	228
Annotating an XML Schema	233
Summary	234

Q&A	236
Workshop	237
Quiz	237
Exercises	237
PART I In Review	239
Well-Formed Documents	241
Valid Documents	241
PART II At a Glance	247
Formatting XML Documents	247
DAY 8 Formatting XML by Using Cascading Style Sheets	249
Our Sample XML Document	250
Introducing CSS	252
Connecting CSS Style Sheets and XML Documents	254
Creating Style Sheet Selectors	256
Creating Style Classes	257
Selecting by ID	261
Using Inline Styles	262
Creating Style Rule Specifications in Style Sheets	263
Creating Block Elements	264
Specifying Text Styles	264
Styling Colors and Backgrounds	265
Styling Borders	268
Styling Alignments	269
Styling Images	270
Positioning Elements	274
Styling Lists	278
Styling Tables	279
Summary	281
Q&A	283
Workshop	283
Quiz	284
Exercises	284
DAY 9 Formatting XML by Using XSLT	285
Introducing XSLT	286
Transforming XML by Using XSLT	288
Server-Side XSLT	288
Client-Side XSLT	290
Standalone Programs and XSLT	291
Writing XSLT Style Sheets	293

Using <xsl:apply-templates>	295
Using <xsl:value-of> and <xsl:for-each>	298
Matching Nodes by Using the match Attribute	301
Handling the Root Node	301
Handling Elements	301
Handling Attributes	302
Handling ID Attributes	306
Handling Processing Instructions	306
Handling Multiple Matches	306
Matching Using XPath Expressions	308
Working with the select Attribute and XPath	309
Using Axes	310
Using Node Tests	311
Using Predicates	311
XPath Abbreviations and Default Rules	317
XPath Tools	321
Using <xsl:copy>	321
Using <xsl:if>	323
Using <xsl:choose>	324
Specifying the Output Document Type	327
Summary	328
Q&A	329
Workshop	329
Quiz	330
Exercises	330
Day 10 Working with XSL Formatting Objects	331
Introducing XSL-FO	332
Using XSL-FO	333
Using XSLT to Create an XSL-FO Document	335
Creating an XSL-FO Document by Using an XSLT Style Sheet	337
Creating a PDF Document	339
Using XSL Formatting Objects and Properties	341
Building an XSL-FO Document	344
Using <fo:root>	344
Using <fo:layout-master-set>	345
Using <fo:simple-page-master>	345
Using <fo:region-body> and <fo:region-after>	347
Using <fo:page-sequence>	348
Using <fo:flow>	349
Using <fo:block>	350

Handling Inline Formatting	353
Using <fo:inline>	354
Using <fo:external-graphic>	357
Using <fo:page-number>	360
Formatting Lists	362
Formatting Tables	365
Summary	368
Q&A	370
Workshop	370
Quiz	370
Exercises	370
PART II In Review	371
Using CSS	371
Using XSLT	373
Using XSL-FO	375
PART III At a Glance	379
XML at Work	379
DAY 11 Extending HTML with XHTML	381
Why XHTML?	382
Introducing XHTML 1.0	384
Introducing XHTML 1.1	385
Introducing XHTML 2.0	385
Introducing XHTML Basic	386
Writing XHTML Documents	386
Dissecting the Example	387
Validating XHTML Documents	390
The Basic XHTML Elements	391
Using the Document Element: <html>	391
Creating a Document Head: <head>	392
Giving a Document a Title: <title>	393
Giving a Document a Body: <body>	394
Organizing Text	397
Creating Paragraphs: <p>	398
Skipping a Line: 	400
Centering Text: <center>	400
Styling Block Content: <div>	402
Styling Inline Content: 	405
Creating Headings: <h1> to <h6>	406

Formatting Text	408
Using Bold on Text: <code></code>	408
Italicizing Text: <code><i></code>	410
Underlining Text: <code><u></code>	411
Selecting Fonts: <code></code>	411
Comments: <code><!--></code>	415
Summary	415
Q&A	416
Workshop	417
Quiz	417
Exercises	417
Day 12 Putting XHTML to Work	419
Creating Hyperlinks: <code><a></code>	419
Linking to Other Documents: <code><link></code>	422
Handling Images: <code></code>	425
Creating Frame Documents: <code><frameset></code>	427
Creating Frames: <code><frame></code>	429
Creating Embedded Style Sheets: <code><style></code>	432
Formatting Tables: <code><table></code>	435
Creating Table Rows: <code><tr></code>	437
Formatting Table Headers: <code><th></code>	438
Formatting Table Data: <code><td></code>	440
Extending XHTML	443
Summary	446
Q&A	448
Workshop	448
Quiz	448
Exercises	448
Day 13 Creating Graphics and Multimedia: SVG and SMIL	449
Introducing SVG	450
Creating an SVG Document	454
Creating Rectangles	456
Adobe's SVG Viewer	456
Using CSS Styles	457
Creating Circles	460
Creating Ellipses	461
Creating Lines	462
Creating Polylines	462
Creating Polygons	463
Creating Text	464
Creating Gradients	466

Creating Paths	467
Creating Text Paths	469
Creating Groups and Transformations	471
Creating Animation	472
Creating Links	474
Creating Scripts	476
Embedding SVG in HTML	478
Introducing SMIL	479
Summary	482
Q&A	483
Workshop	483
Quiz	484
Exercises	484
Day 14 Handling XLinks, XPointers, and XForms	485
Introducing XLinks	486
Using xlink:type	491
Using xlink:href	491
Using xlink:show	492
Using xlink:actuate	493
Using xlink:role and xlink:title	493
Using xlink:arcrole and xlink:label	493
Beyond Simple XLinks	494
Creating Arcs	495
Creating Linkbases	497
Introducing XPointers	498
Using Barenames	499
Using the Element Scheme	499
Using the Namespace Scheme	500
Using the XPointer Scheme	500
Creating XPointer Points	502
Creating XPointer Ranges	503
Introducing XBase	504
Introducing XForms	504
Writing XForms	509
Separating Data from a Presentation	510
Creating Input Controls	512
Creating Select Controls	512
Creating Buttons	513
Creating Select Booleans	515
Creating Submit and Reset Buttons	516
Summary	517
Q&A	518

Workshop	518
Quiz	518
Exercises	519
PART III In Review	521
PART IV At a Glance	529
Programming and XML	529
DAY 15 Using JavaScript and XML	531
Introducing the W3C DOM	532
The DOM Levels	533
Introducing the DOM Objects	534
Using the DOMDocument Object	536
Using the XMLDOMNode Object	539
Using the XMLDOMElement Object	540
Using the XMLDOMAttribute Object	541
Using the XMLDOMText Object	542
Working with the XML DOM in JavaScript	544
Searching for Elements by Name	549
Reading Attribute Values	551
Getting All XML Data from a Document	554
Validating XML Documents by Using DTDs	560
Summary	564
Q&A	565
Workshop	566
Quiz	566
Exercises	566
DAY 16 Using Java and .NET: DOM	567
Using Java to Read XML Data	568
Looping Over Nodes	574
Handling Document Nodes	576
Handling Elements	577
Handling Attributes	577
Handling Child Nodes	579
Handling Text Nodes	579
Handling Processing Instructions	580
Handling CDATA Sections	580
Ending Elements	580
Finding Elements by Name	584
Creating an XML Browser by Using Java	589
Navigating Through XML Documents	596
Writing XML by Using Java	597

Summary	604
Q&A	605
Workshop	605
Quiz	606
Exercises	606
Day 17 Using Java and .NET: SAX	607
An Overview of SAX	608
Using SAX	610
Handling the Start of a Document	614
Handling Processing Instructions	615
Handling the Start of an Element	615
Handling Attributes	616
Handling Text	618
Handling the End of Elements	619
Handling Errors and Warnings	619
Using SAX to Find Elements by Name	623
Creating an XML Browser by Using Java and SAX	628
Navigating Through XML Documents by Using SAX	633
Writing XML by Using Java and SAX	637
Summary	641
Q&A	642
Workshop	642
Quiz	642
Exercises	643
Day 18 Working with SOAP and RDF	645
Introducing SOAP	646
Understanding SOAP Syntax	646
Introducing the SOAP Elements	647
Introducing the SOAP Attributes	649
A SOAP Example in .NET	650
Creating a SOAP Server	651
Creating a SOAP Client	653
Using the Server and Client	655
A SOAP Example in Java	656
Creating the Server	658
Creating the Client	662
Introducing RDF	668
Understanding How RDF Documents Work	670
Creating RDF Root Elements	671
Creating Description Elements	672
Creating Property Elements	672

Using the Dublin Core	673
Working with Multiple Resources	675
Using Resource Attributes	677
Using XML in RDF Elements	678
Using Abbreviated RDF	679
Summary	680
Q&A	681
Workshop	681
Quiz	681
Exercises	682
PART IV In Review	683
PART V At a Glance	691
Data Handling and XML	691
DAY 19 Handling XML Data Binding	693
Introducing DSOs	694
Binding HTML Elements to HTML Data	696
Binding HTML Elements to XML Data	703
Binding HTML Tables to XML Data	706
Accessing Individual Data Fields	709
Binding HTML Elements to XML Data by Using the XML DSO	711
Binding HTML Tables to XML Data by Using the XML DSO	714
Searching XML Data by Using a DSO and JavaScript	716
Handling Hierarchical XML Data	720
Summary	725
Q&A	726
Workshop	726
Quiz	726
Exercises	726
DAY 20 Working with XML and Databases	727
XML, Databases, and ASP	728
Storing Databases as XML	731
Using XPath with a Database	743
Introducing XQuery	749
Summary	761
Q&A	762
Workshop	762
Quiz	762
Exercises	763

DAY 21 Handling XML in .NET	765
Creating and Editing an XML Document in .NET	766
Creating a New XML Document in .NET	766
Creating a Simple Type in an XML Schema in .NET	768
Creating a Complex Type in an XML Schema	770
Creating an Element	771
Creating a Document Element	772
Connecting an XML Schema to an XML Document	773
Working With XML Data	774
From XML to Databases and Back	776
Reading and Writing XML in .NET Code	777
Writing XML in .NET	777
Reading XML	780
Using XML Controls to Display Formatted XML	784
Creating XML Web Services	789
Creating a Web Service	790
Using a Web Service	792
Summary	797
Q&A	798
Workshop	798
Quiz	798
Exercises	799
PART V In Review	801
APPENDIX A Quiz Answers	807
Quiz Answers for Day 1	807
Quiz Answers for Day 2	808
Quiz Answers for Day 3	808
Quiz Answers for Day 4	808
Quiz Answers for Day 5	809
Quiz Answers for Day 6	809
Quiz Answers for Day 7	810
Quiz Answers for Day 8	810
Quiz Answers for Day 9	810
Quiz Answers for Day 10	811
Quiz Answers for Day 11	811
Quiz Answers for Day 12	811
Quiz Answers for Day 13	812
Quiz Answers for Day 14	812

Quiz Answers for Day 15	812
Quiz Answers for Day 16	813
Quiz Answers for Day 17	813
Quiz Answers for Day 18	814
Quiz Answers for Day 19	814
Quiz Answers for Day 20	814
Quiz Answers for Day 21	815

Index

817

About the Author

Steven Holzner is an award-winning author who has written 80 computing books. He has been writing about XML since it first appeared and is one of the foremost XML experts in the United States, having written several XML bestsellers and being a much-requested speaker on the topic. He's also been a contributing editor at *PC Magazine*, has been on the faculty of Cornell University and MIT, and teaches corporate programming classes around the United States.

Dedication

To Nancy, as always and forever—for all the reasons she already knows!

Acknowledgments

A book like the one you're reading is the product of many people's hard work. I'd especially like to thank Todd Green, the acquisitions editor; Songlin Qiu, the development editor; Matt Purcell, the project editor; and Christian Kenyeres, the tech editor.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you would like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an associate publisher for Sams Publishing, I welcome your comments. You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@sampublishing.com

Mail: Michael Stephens
Associate Publisher
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

For more information about this book or another Sams Publishing title, visit our Web site at <http://www.sampublishing.com>. Type the ISBN (0672325764) or the title of a book in the Search field to find the page you're looking for.

This page intentionally left blank

Introduction

Welcome to Extensible Markup Language (XML), the most influential innovation the Internet has seen in years. XML is a powerful, very dynamic topic, spanning dozens of fields, from the simple to the very complex. This book opens up that world, going after XML with dozens of topics—and hundreds of examples.

Unlike other XML books, this book makes it a point to show how XML actually works, making sure that you see everything demonstrated with examples. The biggest problem with most XML books is that they discuss XML and its allied specifications in the abstract, which makes it very hard to understand what's going on. This book, however, illustrates every XML discussion with examples. It shows all that's in the other books and more besides, emphasizing seeing things at work to make it all clear.

Instead of abstract discussions, this book provides concrete working examples because that's the only way to really learn XML. You're going to see where to get a lot of free software on the Internet to run the examples you create—everything from XML browsers to XPath visualizers to XQuery processors to XForms handlers, which you don't find in other books. You'll create XML-based documents that display multimedia shows you can play in RealPlayer, use browser plug-ins to handle XML-based graphics in the popular Hypertext Markup Language (HTML) browsers, enable Web pages to load and handle XML, and much more. XML can get complicated, and seeing it at work is the best way to understand it.

What This Book Covers

This book covers XML as thoroughly as any book you'll find: It goes from the most basic up through the advanced. XML ranges over many disciplines, and this book tracks it down where it lives. Part I, "Creating XML Documents," shows how to use XML in both current Web browsers as well as specialized XML-only browsers. Part I works through every part of an XML document to show how to construct such documents. You'll see how to use online XML validators to check XML and where to find software that lets you check an XML document's schema to make sure the document works as it should. You'll see how to format XML by using cascading style sheets (CSS), Extensible Stylesheet Language Transformations (XSLT), and XML-based formatting objects.

You don't need any programming skills to work with XML in Part I of this book. However, there's no way to ignore the terrific amount of XML support in programming languages such as JavaScript, Java, and the .NET programming languages. Later in the

book, you'll see how to use those languages with XML, navigating through XML documents, extracting data, formatting data, and even creating your own simple XML browsers.

Here's an overview of some of the topics covered in this book:

- The basics of XML
- Displaying XML in browsers
- Writing XML
- Creating well-formed and valid XML documents
- Working with XML validators
- Finding XML resources on the Internet
- Creating Document Type Definitions (DTDs)
- Creating XML schema
- Using XML schema-generating tools
- Using CSS with XML documents
- Displaying images
- Using XSLT to transform XML in the server, in the client, and with standalone programs
- Creating XSLT stylesheets
- Working with XPath
- Using the XSL formatting language
- Introducing Extensible HTML (XHTML)
- Validating XHTML
- Drawing basic shapes in Scalable Vector Graphics (SVG)
- Using SVG hyperlinks, animation, scripting, and gradients
- Creating SMIL documents
- Using Synchronized Multimedia Integration Language (SMIL)
- Creating XLinks, XPointers, and XForms
- Separating data and presentations in XForms
- Handling XML with JavaScript
- Using Java and the XML Document Object Model (DOM)
- Using XML data islands
- Parsing XML documents

- Navigating through an XML document by using Java
- Creating graphical XML browsers by using Java
- Using Java and the Simple API for XML (SAX)
- Using Simple Object Access Protocol (SOAP) to communicate between Web applications
- Binding XML data to HTML controls
- Navigating through XML data
- Displaying XML data in tables
- Managing XML databases
- Working with XML database storage in .NET
- Using XQuery to query an XML document
- Editing XML documents and XML schemas in .NET
- Writing and reading XML documents from code
- Creating XML Web services

As you can see, this book covers many facets of XML.

Who This Book Is For

This book is for anyone who wants to learn XML and how it is used today. This book assumes that you've had some experience with HTML, but that's about all it assumes. In Part IV, "Programming and XML," knowledge of JavaScript and Java helps, although the chapters in Part IV discuss where you can find free online tutorials on these subjects. The .NET programming discussed on Day 21, "Handling XML in .NET," may be a little hard to follow unless you've worked with Visual Basic .NET before.

Note that this book is as platform-independent as possible. XML is not the province of any one particular operating system, so this book does not lean one way or another on that issue. This book aims to show you as much of XML as it can, in the greatest depth possible. However, it's a fact of life that a great deal of XML software these days is targeted at Windows. And among the standard browsers, Internet Explorer has many times more XML support than any other browser does. This book doesn't have any special pro- or anti-Microsoft bias, but in order for this book to cover what's available for XML these days, you're going to find yourself in Microsoft territory fairly often; there's no getting around it.

Conventions Used in This Book

The following conventions are used in this book:

- Code lines, commands, statements, and any other code-related terms appear in a monospace typeface. Placeholders (which stand for what you should actually type) appear in *italic monospace*. Text that you should type appears in **bold**.
- When a line of code is too long to fit on one line of this book, it is broken at a convenient place and continued to the next line. The continuation is preceded by a special code continuation character (➞).
- New lines of XML or programming code that are added and are being discussed appear shaded, and when there's more code to come, you see three vertical dots. Here's how these features look:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    .
    .
    .
</document>
```

- Throughout the book are notes that are meant to give you something more. This is what a note looks like:

NOTE

A note presents interesting information related to the discussion—a little more insight or a pointer to some new technique.

- This book also contains tips. This is what a tip looks like:

TIP

A tip offers advice or shows you an easier way of doing something.

- This book also contains cautions. This is what a caution looks like:

CAUTION

A caution alerts you to a possible problem and gives you advice on how to avoid it.

- Each day's lesson ends with questions pertaining to that day's subject matter, with answers from the book's author. Each day's discussion also includes a quiz that is designed to test your knowledge of the day's concepts. The answers to these quiz questions are provided in Appendix A, "Answers to Quiz Questions." Many lessons conclude with exercises that give you an opportunity to practice what you've learned in the lesson.

Where to Download the Book's Code

You can download all the code examples used throughout this book from <http://www.sampublishing.com>. Simply enter this book's ISBN without the hyphens (**0672325764**) in the Search box and click Search. When the book's title is displayed, click it to go to a page where you can download the code.

This page intentionally left blank

PART I

DAY 3

Creating Well-Formed XML Documents

Yesterday, you took a look at the various parts of XML documents—prologs, elements and attributes, processing instructions, and so forth. Today, you’re going to start putting those items to work as you create well-formed documents.

Why is it so important to make an XML document well-formed? For one thing, W3C doesn’t consider an XML document to be XML unless it’s well-formed. For another, XML processors won’t read XML documents unless those documents are well-formed. All of which is to say that making your XML well-formed is integral to creating XML documents—software isn’t even going to be able to read your documents unless they are. Here’s an overview of today’s topics:

- Well-formed XML documents
- The W3C Well-formedness constraints
- Nesting constraints
- Element and attribute constraints

- Namespaces
- Local and default namespaces
- XML Infosets
- Canonical XML

To some extent, the current loose state of HTML documents is responsible for the great emphasis W3C puts on making sure XML documents are well-formed. HTML browsers have become more and more friendly to HTML pages as time has gone on, which means a Web page can have dozens of errors and still be displayed by a browser. That's not such a problem when it comes to simply displaying a Web page, but when it comes to handling what might be crucial data, it's a different story.

So W3C changed the rules from HTML to XML—unlike an HTML browser, an XML processor is *never* supposed to guess when it reads an XML document. If it finds an error (if the document is not well-formed, or if it uses a DTD or XML schema and it's not valid), the XML processor is supposed to inform you of the error, but then it can quit immediately. Ideally, according to W3C, a validating XML processor should list all the errors in an XML document and then quit; a non-validating one doesn't even have to do that—it can quit the first time it sees an error.

This enforced precision has two sides to it—there's no doubt that your data is transferred more faithfully using XML, but because XML processors make no guesses as to what you're trying to do, XML and XML processors can come across as non-user friendly, and not as generous or as easy to work with as HTML. On the other hand, you don't end up with the many possible errors that can creep into HTML, and that's important. XML authors have to be aware of the constraints on what they write, which is why we spend time in this book on document well-formedness and validity. In fact, in the XML 1.0 specification, W3C says that you can't even call a data object an XML document unless it's well-formed:

A data object is an XML document if it is well-formed, as defined in this specification. A well-formed XML document may in addition be valid if it meets certain further constraints.

What Makes an XML Document Well-Formed?

The W3C, which is responsible for the term *well-formedness*, defines it this way in the XML 1.0 recommendation:

A textual object is a well-formed XML document if:

- Taken as a whole, it matches the production labeled *document*.
- It meets all the well-formedness constraints given in this specification (that is, the XML 1.0 specification, <http://www.w3.org/TR/REC-xml>).
- Each of the parsed entities, which is referenced directly or indirectly within the document, is well-formed.

Because the major differences between XML 1.0 and XML 1.1 have to do with what characters are legal, you probably won't be surprised to learn that a well-formed XML 1.0 document is also a well-formed XML 1.1 document, as long as it avoids certain characters. From the XML 1.1 specification:

If a document is well-formed or valid XML 1.0, and provided it does not contain any characters in the range [#x7F-#x9F] other than as character escapes, it may be made well-formed or valid XML 1.1 respectively simply by changing the version number.

Let's get into three conditions that make an XML document well-formed, starting with the requirement that the document must match the production named *document*.

Matching the Production Labeled *document*

W3C calls the individual specifications within a working draft or recommendation *productions*. In this case, to be well-formed, a document must follow the document production, which means that the document itself must have three parts:

- a prolog (which can be empty)
- a root element (which can contain other elements)
- a miscellaneous part (unlike the preceding two parts, this part is optional)

You've seen XML prologs yesterday; they can contain an XML declaration (such as `<?xml version = "1.0" ?>`), as well as comments, processing instructions, and doctype declarations (that is, DTDs).

You've also seen root elements; the root element is the XML element that contains all the other elements in your document. Each well-formed XML document must have one, and only one, root element.

The optional miscellaneous part can be made up of XML comments, processing instructions, and whitespace, all items you saw yesterday.

In other words, this first requirement says that an XML document must be made up of the parts you saw yesterday. So far, so good.

Meeting the Well-Formedness Constraints

The next requirement is a little more difficult to track down, because it says that to be well-formed, XML documents must also satisfy the well-formedness constraints in the XML 1.0 specification. This means that your XML documents should adhere to the syntax rules specified in the XML 1.0 recommendation. You'll discuss those rules, which are sprinkled throughout the XML 1.0 specification, in a few pages.

Making Parsed Entity Must Be Well-Formed

The final requirement is that each parsed entity in a well-formed document must itself be well-formed. When an XML document is parsed by an XML processor, entity references (such as `π`) are replaced by the entities they stand for (such as Π in this case). The requirement that all parsed entities must be well-formed simply means that when you replace entity references with the entities they stand for, the result must be well-formed.

That's the W3C's definition of a well-formed document, but you still need more information. What are the well-formedness constraints given throughout the XML specification? You're going to go over these constraints today; to start, you'll create an XML document that you'll use as we discuss what it means for a document to be well-formed.

Creating an Example XML Document

The sample document you'll use today, and which you'll also see tomorrow when working with DTDs, will store data about a set of employees, such as their names, projects they're working on, and so on. This document will start, as all XML documents should, with an XML declaration:

```
<?xml version = "1.0"?>
```

Because all the documents you'll see today are self-contained (they don't refer to or include any external entities), you'll also add the `standalone` attribute, setting it to "yes", and specify that we're using UTF-8 encoding:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
```

And you'll also add a root element, called `<document>` in this case, although you can use any legal name:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
  .
  .
  .
</document>
```

The root element will contain all the other elements in the document. In this case, that will be three `<employee>` elements:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
  <employee>
    .
    .
    .
  </employee>
  <employee>
    .
    .
    .
  </employee>
  <employee>
    .
    .
    .
  </employee>
</document>
```

For each employee, we can store a name in a `<name>` element, which itself encloses a `<lastname>` and `<firstname>` element:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
  <employee>
    <name>
      <lastname>Kelly</lastname>
      <firstname>Grace</firstname>
    </name>
    .
    .
    .
  </employee>
  .
  .
  .
</document>
```

We'll also store each employee's hire date, as well as the projects they're working on. For each project, we can store the product name, ID, and price:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
  <employee>
    <name>
      <lastname>Kelly</lastname>
      <firstname>Grace</firstname>
    </name>
```

```

    <hiredate>October 15, 2005</hiredate>
    <projects>
      <project>
        <product>Printer</product>
        <id>111</id>
        <price>$111.00</price>
      </project>
      <project>
        <product>Laptop</product>
        <id>222</id>
        <price>$989.00</price>
      </project>
    </projects>
  </employee>
  .
  .
  .
</document>

```

That's what the data looks like for one employee; you can see the full document, `ch03_01.xml`, in Listing 3.1. Documents like this one can grow very long, but that presents no problem to XML processors—as long as the document is well-formed.

LISTING 3.1 Sample Well-Formed XML Document (`ch03_01.xml`)

```

<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
  <employee>
    <name>
      <lastname>Kelly</lastname>
      <firstname>Grace</firstname>
    </name>
    <hiredate>October 15, 2005</hiredate>
    <projects>
      <project>
        <product>Printer</product>
        <id>111</id>
        <price>$111.00</price>
      </project>
      <project>
        <product>Laptop</product>
        <id>222</id>
        <price>$989.00</price>
      </project>
    </projects>
  </employee>
  <employee>
    <name>
      <lastname>Grant</lastname>

```

LISTING 3.1 continued

```
<firstname>Cary</firstname>
</name>
<hiredate>October 20, 2005</hiredate>
<projects>
  <project>
    <product>Desktop</product>
    <id>333</id>
    <price>$2995.00</price>
  </project>
  <project>
    <product>Scanner</product>
    <id>444</id>
    <price>$200.00</price>
  </project>
</projects>
</employee>
<employee>
  <name>
    <lastname>Gable</lastname>
    <firstname>Clark</firstname>
  </name>
  <hiredate>October 25, 2005</hiredate>
  <projects>
    <project>
      <product>Keyboard</product>
      <id>555</id>
      <price>$129.00</price>
    </project>
    <project>
      <product>Mouse</product>
      <id>666</id>
      <price>$25.00</price>
    </project>
  </projects>
</employee>
</document>
```

Today's work gets us into the structure of XML documents, and there's some terminology we should get to know at this point having to do with the relative position of elements in an XML document. As an example, take a look at an employee element in `ch03_01.xml`.

Elements on the same level, such as `<name>`, `<hiredate>`, and `<projects>` in an `<employee>` element, are all called *siblings*. Similarly, the two `<project>` elements in each `<projects>` element are siblings.

This family-type relationship is also continued with *child* and *parent* relationships. For example, the parent of the two `<project>` elements is the `<projects>` element. And the two `<project>` elements are children of the `<projects>` element.

You can always count on every non-root element to have exactly one, and only one, parent element. And a parent element can enclose an indefinite number of child elements (which can also mean zero child elements). You can continue the analogy to multiple generations as well; for example, the two `<project>` elements in this case are also *grandchildren* of the `<employee>` element.

That gives us the example document and terminology we'll need; now let's take a look at the well-formedness constraints you'll find in XML.

Understanding the Well-Formedness Constraints

The well-formedness constraints in the XML 1.0 specification are sprinkled throughout the document, and some of them are hard to dig out because they're not clearly marked. You'll get a look at the well-formedness constraints here, although note that some of them have to do with DTDs and entity references, and those will appear in Day 4, "Creating Valid XML Documents: Document Type Definitions," and Day 5, "Handling Attributes and Entities in DTDs."

Beginning the Document with an XML Declaration

The first well-formedness structure constraint is to start the document with an XML declaration. Even though some XML processors won't insist on it, W3C says you should always include this declaration first thing:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
  <employee>
    .
    .
    .
```

TIP

Although the XML 1.0 specification says that only the `version` attribute is required here, some software—notably including W3C's own Amaya testbed browser—will consider XML documents as not well-formed if you don't also include the `encoding` attribute.

Using Only Legal Character References

Another well-formedness constraint is that character references, which are character codes enclosed in & and ;, and which are replaced by the characters that code stands for, must only refer to characters supported by the XML specification.

This constraint is more or less obvious—it simply means that you have to stick to the established character set for the version of XML you’re using. Note that, as you saw yesterday, the characters that are legal in XML 1.0 differ somewhat from what’s legal in XML 1.1.

Including at Least One Element

To be a well-formed document, a document must include *one or more* elements. The first element, of course, is the root element, so to be well-formed, a document must contain at least a root element. In other words, an XML document must contain more than just a prolog. Of course, your documents will usually contain many elements, as in our example document:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
  <employee>
    <name>
      <lastname>Kelly</lastname>
      <firstname>Grace</firstname>
    </name>
    <hiredate>October 15, 2005</hiredate>
    <projects>
      <project>
        .
        .
        .
      </project>
    </projects>
  </employee>
  .
  .
</document>
```

Structuring Elements Correctly

HTML browsers are pretty easygoing about how you structure HTML elements in a Web page as long as they can understand what you’re doing. For example, you can often omit closing tags in elements—you might use a <p> tag and then follow it with another <p> tag—without using a </p> tag—and the browser will have no problem.

That's not the way things work in XML. In XML, every non-empty element must have both a start tag and an end tag, as in our example document:

```
<employee>
  <name>
    <lastname>Gable</lastname>
    <firstname>Clark</firstname>
  </name>
  <hiredate>October 25, 2005</hiredate>
  <projects>
    <project>
      <product>Keyboard</product>
      <id>555</id>
      <price>$129.00</price>
    </project>
    <project>
      <product>Mouse</product>
      <id>666</id>
      <price>$25.00</price>
    </project>
  </projects>
</employee>
```

Besides making sure that every non-empty element has an opening tag and a closing tag, another well-formedness constraint says that end tags must match start tags, and both must use the same name.

Some elements—empty elements—don't have closing tags. These tags have no content of any kind (although they can have attributes), which means that they do not enclose any character data or markup. Instead, these elements are made up entirely of one tag like this:

```
<?xml version = "1.0" standalone="yes"?>
<document>
  <heading text = "Hello From XML" />
</document>
```

In XML, empty elements must always end with />.

TIP

HTML elements can also be ended with />, such as
, and HTML browsers will not have a problem with them. That's good, because the alternative is to write
</BR>, which some browsers, such as Netscape Navigator, interpret as two
 elements.

Using the Root Element to Contain All Other Elements

Another well-formedness constraint is that the root element must contain all the other elements in the document, as in our sample XML document, where we have three `<employee>` elements, which themselves contain other elements, in the document element:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
  <employee>
    .
    .
    .
  </employee>
  <employee>
    .
    .
    .
  </employee>
  <employee>
    .
    .
    .
  </employee>
</document>
```

That's how a well-formed XML document works—you start with a prolog, followed by the root element, which contains all the other the elements, if there are any. Among other things, containing all elements in a root element makes it easier for an XML processor to understand the structure of an XML document—starting at the single root element, it can navigate the entire document.

Nesting Elements Properly

Nesting elements correctly is a big part of well-formedness; the requirement here is that if an element contains a start tag for a non-empty tag, it must also contain that element's end tag. In other words, you cannot spread an element over other elements at the same level. For example, this XML is nested properly:

```
<employee>
  <name>
    <lastname>Kelly</lastname>
    <firstname>Grace</firstname>
  </name>
  <hiredate>October 15, 2005</hiredate>
  <projects>
    <project>
      <product>Printer</product>
      <id>111</id>
    </project>
  </projects>
</employee>
```

```

        <price>$111.00</price>
    </project>
    <project>
        <product>Laptop</product>
        <id>222</id>
        <price>$989.00</price>
    </project>
</projects>
</employee>

```

But as you can see, there's a nesting problem in this next element, because an XML processor will encounter a new `<project>` tag before finding the closing `</project>` tag it's looking for at the end of the current `<project>` element:

```

<employee>
    <name>
        <lastname>Kelly</lastname>
        <firstname>Grace</firstname>
    </name>
    <hiredate>October 15, 2005</hiredate>
    <projects>
        <project>
            <product>Printer</product>
            <id>111</id>
            <price>$111.00</price>
        <project>
        </project>
            <product>Laptop</product>
            <id>222</id>
            <price>$989.00</price>
        </project>
    </projects>
</employee>

```

In fact, this nesting requirement is where the whole term *well-formed* comes from—the original idea was that a document where the elements were not garbled and mixed up with each other was well-formed.

There are other well-formedness constraints that have nothing to do with elements, however—for example, the next two concern attributes.

Making Attribute Names Unique

Another well-formedness constraint is that you can't use the same attribute more than once in one start-tag or empty-element tag. This is another well-formedness constraint that seems more or less obvious, and it's hard to see how you might violate this one except by mistake, as in this case:

```
<message text="Hi there!" text="Hello!">
```

XML is case sensitive, so you could theoretically do something like this:

```
<message Text="Hi there!" text="Hello!">
```

Obviously, that's not a very good idea, however; attribute names that differ only in capitalization are bound to be confusing.

Enclose Attribute Values in Quotation Marks

One well-formedness constraint that trips up most XML novices sooner or later is that you must quote every value you assign to an attribute, using either single quotation marks or double quotation marks. This trips many people up because you don't have to quote attribute values in HTML, as in this HTML example (which also doesn't have a closing tag):

```
<img src=mountains.jpg>
```

An XML processor would have problems with this element, however. Here's what it would look like properly constructed:

```

```

If you prefer, you could use single quotation marks:

```
<img src='mountains.jpg' />
```

As you've seen, using single quotation marks helps when an attribute's value contains quoted text:

```
<message text='I said, "No, no, no!"' />
```

And as you've also seen, in worst-case scenarios, where an attribute value contains both single and double quotation marks, you can escape " as "; and ' as ';—as here, where you're reporting the height of a tree as 50' 6" :

```
<tree type="Maple" height="50&apos;6&quot;" />
```

Avoiding Entity References and < in Attribute Values

Also, W3C makes it an explicit well-formedness constraint that you should avoid references to external entities (this means XML-style references—general entity references or parameter entity references, not just, for example, using an image file's name) in attribute values. This means that an XML processor doesn't have to replace an attribute value with the contents of an external entity.

In addition, another constraint says that you are not supposed to use < in attribute values, because an XML processor might mistake it for markup. If you really have to use the text <, use < instead, which will be turned into < when parsed. For example, this XML:

```
<project note="This is a <project> element.">
```

should be written as this, where you're escaping both < and >:

```
<project note="This is a &lt;project&gt; element.">
```

In fact, < is a particularly sensitive character to use anywhere in an XML document, except as markup, and that's another well-formedness constraint concerning <, coming up next.

Avoiding Overuse of < and &

XML processors assume that < starts a tag and & starts an entity reference, so you should avoid using those characters for anything else. Sometimes, this is a problem, as in the JavaScript example you saw yesterday, which uses the JavaScript < operator that enclosed in a CDATA section:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/tr/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>
      Checking the temperature
    </title>
  </head>

  <body>
    <script language="javascript">
      <![CDATA[
        var temperature
        temperature = 234.77
        if (temperature < 32) {
          document.writeln("Below freezing!")
        }
      ]]>
    </script>

    <center>
      <h1>
        Checking the temperature
      </h1>
    </center>
  </body>
</html>
```

However, because modern Web browsers don't understand CDATA sections, this solution (which was suggested by W3C) doesn't really work. And if you escape the > operator as < , very few browsers will understand what you're doing.

There are two main ways of handling the < JavaScript operator in XML with today's browsers. You can reverse the logical sense of the test—for example, in this case, instead of checking whether the temperature is below 32, you would check to make sure it isn't above or equal to 32, which lets you use > instead of < (note that the JavaScript ! operator, the Not operator, reverses the logical sense of an expression) :

```
<script language="javascript">
  var temperature
  temperature = 234.77
  if (!(temperature >= 32)) {
    document.writeln("Below freezing!")
  }
</script>
```

Practically speaking, the best way is usually to remove the whole problem by placing the script code in an external file, which you'll name `script.js` here, so the browser won't parse it as XML in the first place. You can do that like this in JavaScript (more on JavaScript and how to use it in XML is coming up in Day 15, "Using JavaScript and XML"):

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/tr/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>
      Checking the temperature
    </title>
  </head>

  <body>
    <script language="javascript" src="script.js">
    </script>
    <center>
      <h1>
        Checking the temperature
      </h1>
    </center>
  </body>
</html>
```

That completes today's discussion of well-formedness, although you'll see more in the next two days as we discuss the well-formedness constraints that have to do with DTDs.

As your XML documents evolve and become more complex, it's also going to be increasingly important to understand namespaces, which are the second major topic for today.

Using XML Namespaces

There's a lot of freedom in XML, because you get to create your own markup. As time went on, however, XML authors started noticing a problem that the original creators of XML hadn't really anticipated—conflicting tag names.

For example, you've already seen that two popular XML applications are XHTML, which is the derivation of HTML in XML, and MathML, which lets you format and display math equations. Suppose that you want to display an equation in an XHTML Web page. That could be a problem, because the tag set in XHTML and MathML overlap—in particular, each XML application defines a `<var>` and `<select>` element.

The way to solve this problem is to use *namespaces*. Namespaces give you a way to make sure that one set of tags will not conflict with another. You prefix a name to tag and attribute names. Changing the resulting names won't conflict with others that have a different prefix.

XML namespaces are one of those XML companion recommendations that keep being added to the XML specification. You can find the specification for namespaces at <http://www.w3.org/TR/REC-xml-names/>. There's still a lot of debate about this one (mostly because namespaces can make writing DTDs difficult), but it's an official W3C recommendation now.

Creating Namespaces

An example will make namespaces and why they're important clearer. For example, suppose you're the boss of one of the employees in our sample document, `ch03_01.xml`:

```
<employee>
  <name>
    <lastname>Kelly</lastname>
    <firstname>Grace</firstname>
  </name>
  <hiredate>October 15, 2005</hiredate>
  <projects>
    <project>
      <product>Printer</product>
      <id>111</id>
      <price>$111.00</price>
    </project>
  </project>
```

```

        <product>Laptop</product>
        <id>222</id>
        <price>$989.00</price>
    </project>
</projects>
</employee>

```

Now suppose that you want to add your own comments to this employee's data in a `<comment>` element. The problem with that is that the XML data on this employee comes from the Human Resources department, and they haven't created an element named `<comment>`. You can indeed create your own `<comment>` element, but first you should confine the human resource's department's XML data to its own namespace to indicate that your comments are not part of the Human Resource Department's set of XML tags.

To define a new namespace, use the `xmlns:prefix` attribute, where *prefix* is the prefix you want to use for the namespace. In this case, you'll define a new namespace called `hr` for the Human Resources department:

```

<employee
  xmlns:hr="http://www.superduperbigco.com/human_resources">
  <name>
    <lastname>Kelly</lastname>
    <firstname>Grace</firstname>
  </name>
  <hiredate>October 15, 2005</hiredate>
  <projects>
    <project>
      <product>Printer</product>
      <id>111</id>
      <price>$111.00</price>
    </project>
    <project>
      <product>Laptop</product>
      <id>222</id>
      <price>$989.00</price>
    </project>
  </projects>
</employee>

```

To define a namespace, you assign the `xmlns:prefix` attribute to a unique identifier, which in XML is usually a URI that might direct the XML processor to a DTD for the namespace (but doesn't have to). So what's a URI?

Defining Namespaces with URIs

The XML specification expands the idea of standard URLs (Uniform Resource Locators) into *URIs (Uniform Resource Identifiers)*. In HTML and on the Web, you use URLs; in

XML, you use URIs. URIs are supposed to be more general than URLs, as we'll see when we discuss XLinks and XPointers in Day 14, "Handling XLinks, XPointers, and XForms."

For example, in theory, a URI can point not just to a single resource, but to a cluster of resources, or to *arcs* of resources along a path. The truth is that the whole idea of URIs as the next step after URLs is still being developed, and in practice, URLs are almost invariably used in XML—but you still call them URIs. Some software accepts more general forms of URIs, letting you, for example, access only a specific section of an XML document, but such usage and the associated syntax is far from standardized yet.

TIP

You might want to look up the current formal definition of URIs, which you can find in its entirety at <http://www.ics.uci.edu/pub/ietf/uri/rfc2396.txt>.

When you define a namespace with the `xmlns:prefix` attribute, you usually assign a URI to that attribute (in practice, this URI is always a URL today). The document that URI points to can describe more about the namespace you're creating; an example of this is the XHTML namespace, which uses the namespace `http://www.w3.org/1999/xhtml`:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/tr/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns:xhtml="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  .
  .
  .
```

A namespace's URI can also hold a DTD or XML schema that defines the syntax for the XML elements you can use in that namespace (then it's up to the XML processor to use that DTD or XML schema, if it's been written to be smart enough to interpret namespaces in this way—most aren't). All that's really necessary, however, is that you assign a unique identifier, which can be any text, to the `xmlns:prefix` attribute.

After defining the `hr` namespace in our example, you can preface every tag and attribute name in this namespace with `hr:` like this:

```
<hr:employee
  xmlns:hr="http://www.superduperbigco.com/human_resources">
  <hr:name>
    <hr:lastname>Kelly</hr:lastname>
    <hr:firstname>Grace</hr:firstname>
  </hr:name>
```

```

<hr:hiredate>October 15, 2005</hr:hiredate>
<hr:projects>
  <hr:project>
    <hr:product>Printer</hr:product>
    <hr:id>111</hr:id>
    <hr:price>$111.00</hr:price>
  </hr:project>
  <hr:project>
    <hr:product>Laptop</hr:product>
    <hr:id>222</hr:id>
    <hr:price>$989.00</hr:price>
  </hr:project>
</hr:projects>
</hr:employee>

```

Now you’ve made it clear that all these tags come from the Human Resources department. Note how this works—the actual tag names themselves have been changed, because a colon is a legal character to use in tag names. (Now you know why you shouldn’t use colons in tag names, although they’re legal—they can make it look like you’re using namespaces when you’re not.) For example, the `<product>` tag has now become the `<hr:product>` tag. In other words, using namespaces keeps elements separate by actually changing tag and attribute names. This was a clever solution to the problem of tag and attribute name conflicts, because this way, even XML processors that have never heard of namespaces can still “support” them.

At this point, all tag and attribute names from the `hr` namespace are in their own namespace, so you can add your own namespace to the document, allowing you to use your own elements without fear of conflict. Since you’re the boss, you might start by defining a new namespace named `boss`:

```

<hr:employee
  xmlns:hr="http://www.superduperbigco.com/human_resources"
  xmlns:boss="http://www.superduperbigco.com/big_boss">
  <hr:name>
    <hr:lastname>Kelly</hr:lastname>
    <hr:firstname>Grace</hr:firstname>
  </hr:name>
  <hr:hiredate>October 15, 2005</hr:hiredate>
  <hr:projects>
    <hr:project>
      <hr:product>Printer</hr:product>
      <hr:id>111</hr:id>
      <hr:price>$111.00</hr:price>
    </hr:project>
    <hr:project>
      <hr:product>Laptop</hr:product>
      <hr:id>222</hr:id>
      <hr:price>$989.00</hr:price>
    </hr:project>
  </hr:projects>
</hr:employee>

```

```

        </hr:project>
    </hr:projects>
</hr:employee>

```

Now you can use the new boss namespace to add your own markup to the document, as you see in Listing 3.2.

LISTING 3.2 XML Document with Namespaces (ch03_02.xml)

```

<hr:employee
  xmlns:hr="http://www.superduperbigco.com/human_resources"
  xmlns:boss="http://www.superduperbigco.com/big_boss">
  <hr:name>
    <hr:lastname>Kelly</hr:lastname>
    <hr:firstname>Grace</hr:firstname>
  </hr:name>
  <hr:hiredate>October 15, 2005</hr:hiredate>
  <boss:comment>Needs much supervision.</boss:comment>
  <hr:projects>
    <hr:project>
      <hr:product>Printer</hr:product>
      <hr:id>111</hr:id>
      <hr:price>$111.00</hr:price>
    </hr:project>
    <hr:project>
      <hr:product>Laptop</hr:product>
      <hr:id>222</hr:id>
      <hr:price>$989.00</hr:price>
    </hr:project>
  </hr:projects>
</hr:employee>

```

You can also add your own attributes in the boss namespace as long as you prefix them with boss: this way:

```

<hr:employee>
  xmlns:hr="http://www.superduperbigco.com/human_resources"
  xmlns:boss="http://www.superduperbigco.com/big_boss">
  <hr:name>
    <hr:lastname>Kelly</hr:lastname>
    <hr:firstname>Grace</hr:firstname>
  </hr:name>
  <hr:hiredate>October 15, 2005</hr:hiredate>
  <boss:comment boss:date="10/15/2006">
    Needs much supervision.
  </boss:comment>
  <hr:projects>
    <hr:project>
      <hr:product>Printer</hr:product>
      <hr:id>111</hr:id>

```

LISTING 3.2 continued

```
<hr:price>$111.00</hr:price>
</hr:project>
<hr:project>
  <hr:product>Laptop</hr:product>
  <hr:id>222</hr:id>
  <hr:price>$989.00</hr:price>
</hr:project>
</hr:projects>
</hr:employee>
```

And that's how namespaces work—you can use them to separate tags, even tags with the same name, so there's no conflict. As you can see, using multiple namespaces in the same document is no problem at all—just use the `xmlns:prefix` attribute in the enclosing element to define the appropriate namespace. In fact, you can use this attribute in child elements to redefine an enclosing namespace, if you want to.

Namespace prefixes are really just text prefixed to (*prepended* is the official term) tag and attribute names. They follow the same rules for naming tags and attributes. For example, in XML 1.0, a namespace name can start with a letter or an underscore. The following characters can include underscores, letters, digits, hyphens, and periods. Note also that although colons are legal in tag names, you can't use a colon in a namespace name, for obvious reasons. Also, there are two namespace names that are reserved: `xml` and `xmlns`.

Creating Local Namespaces

The `xmlns:prefix` attribute can be used in any element, not just the document element. Just bear in mind that this attribute defines a namespace for the current element and any enclosed element, which means you shouldn't use the namespace prefix until you've defined the namespace with an attribute like `xmlns:prefix`.

For example, you can create the `boss:` namespace prefix and use it in the same element, as you see in Listing 3.3.

LISTING 3.3 XML Document with a Local Namespaces (ch03_03.xml)

```
<hr:employee
  xmlns:hr="http://www.superduperbigco.com/human_resources">
  <hr:name>
    <hr:lastname>Kelly</hr:lastname>
    <hr:firstname>Grace</hr:firstname>
  </hr:name>
  <hr:hiredate>October 15, 2005</hr:hiredate>
  <boss:comment
```

LISTING 3.3 continued

```

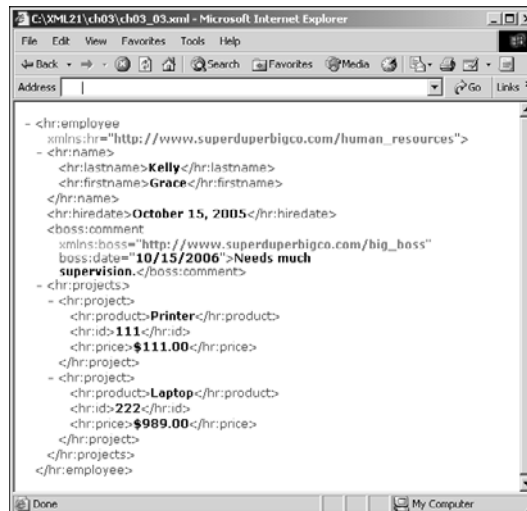
    xmlns:boss="http://www.superduperbigco.com/big_boss"
    boss:date="10/15/2006">
    Needs much supervision.
  </boss:comment>
</hr:projects>
  <hr:project>
    <hr:product>Printer</hr:product>
    <hr:id>111</hr:id>
    <hr:price>$111.00</hr:price>
  </hr:project>
  <hr:project>
    <hr:product>Laptop</hr:product>
    <hr:id>222</hr:id>
    <hr:price>$989.00</hr:price>
  </hr:project>
</hr:projects>
</hr:employee>

```

You can see ch03_03.xml in the Internet Explorer, complete with namespaces, in Figure 3.1.

FIGURE 3.1

Viewing an XML document with local namespaces.



Creating Default Namespaces

You can use the `xmlns:prefix` attribute to define a namespace, or you can use the `xmlns` attribute by itself to define a *default* namespace. When you define a default namespace, elements and attributes without a namespace prefix are in that default namespace.

To see how this works, we'll come full circle and put to work the example that introduced our discussion of namespaces in the first place—mixing XHTML with MathML. We'll start with some XHTML (all the details on XHTML are coming up in Day 11, “Extending HTML with XHTML,” and Day 12, “Putting XHTML to Work”), like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/tr/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>
      Using XHTML and MathML Together
    </title>
  </head>

  <body>
    <center>
      <h1>
        Using XHTML and MathML Together
      </h1>
    </center>
    <br/>
    Consider the equation
    .
    .
    .
  </body>
</html>
```

You'll see what you need to create XHTML documents like this, such as the `<!DOCTYPE>` element, in Day 11. Note in particular here that in the `<html>` element, the `xmlns` attribute defines a default namespace for the `<html>` and all enclosed elements. (This namespace is the XHTML namespace, which W3C defines as `"http://www.w3.org/1999/xhtml"`.) When you use the `xmlns` attribute alone this way, without specifying any prefix, you are defining a default namespace. The current element and all child elements are assumed to belong to that namespace. Making use of a default namespace in this way, you can use the standard XHTML tag names without any prefix, as you see here.

However, we also want to use MathML markup in this document, and to do that, we add a new namespace, named `m` to this document, using the namespace W3C has specified for MathML, `"http://www.w3.org/1998/Math/MathML"`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/tr/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
  xmlns:m="http://www.w3.org/1998/Math/MathML">
  <head>
```

```

        <title>
            Using XHTML and MathML Together
        </title>
    </head>

    <body>
        <center>
            <h1>
                Using XHTML and MathML Together
            </h1>
        </center>
        <br/>
        Consider the equation
        .
        .
        .
    </body>
</html>

```

Now you can use MathML as you like, as long as you prefix it with the `m` namespace. You can see this at work in `ch03_04.html` (XHTML documents use the extension `.html`), shown in Listing 3.4, where we're using the MathML we developed in Day 1 to display an equation.

LISTING 3.4 An XML Document Combining XHTML and MathML (`ch03_04.html`)

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/tr/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
xmlns:m="http://www.w3.org/1998/Math/MathML">
    <head>
        <title>
            Using XHTML and MathML Together
        </title>
    </head>

    <body>
        <center>
            <h1>
                Using XHTML and MathML Together
            </h1>
        </center>
        <br/>
        Consider the equation
        <m:math>
            <m:mrow>
                <m:mrow>
                    <m:mn>4</m:mn>

```

LISTING 3.4 continued

```

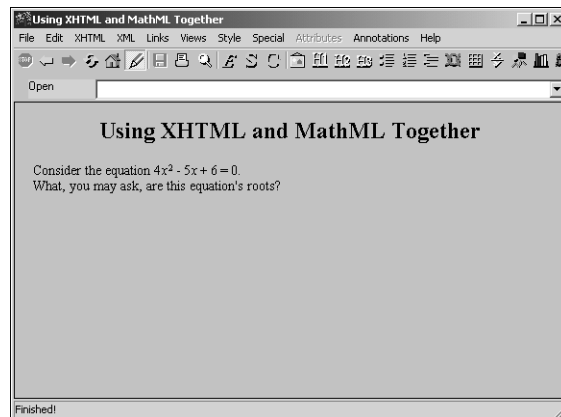
        <m:mo>&InvisibleTimes;</m:mo>
        <m:msup>
            <m:mi>x</m:mi>
            <m:mn>2</m:mn>
        </m:msup>
        <m:mo>-</m:mo>
        <m:mrow>
            <m:mn>5</m:mn>
            <m:mo>&InvisibleTimes;</m:mo>
            <m:mi>x</m:mi>
        </m:mrow>
        <m:mo>+</m:mo>
        <m:mn>6</m:mn>
    </m:mrow>
    <m:mo>=</m:mo>
    <m:mn>0.</m:mn>
</m:mrow>
</m:math>
<br/>
    What, you may ask, are this equation's roots?
</body>
</html>

```

Thanks to namespaces, this XHTML/MathML document works just as it should, as you can see in the W3C Amaya browser in Figure 3.2.

FIGURE 3.2

Viewing an XML document with local namespaces.



You'll be seeing XML namespaces throughout this book, especially when we use the popular XML applications available, such as XHTML.

That finishes the main topics for today's discussion—well-formed documents and namespaces. Before getting into validation in tomorrow's work, however, we'll round off our discussion of XML documents by taking a look at XML infosets and canonical XML. These two topics are worth discussing before we start talking about validation, because they're terms you'll run across as you work with XML, but we're going to consider them optional topics—if you want to skip them and get directly to DTDs, just turn to Day 4.

Understanding XML Infosets

The inspiration behind both *XML infosets* (formally named *XML information sets*) and canonical XML is to make handling the data in XML documents easier. Reducing an XML document down to its infoset is intended to make comparisons between all kinds of XML documents easier by presenting the data in those documents in a standard way. You can find the official XML Information Set specification at <http://www.w3.org/TR/xml-infoset>.

To understand what infosets are and what they're used for, imagine searching for data on the World Wide Web. You might want to search for a particular topic, such as XML, and you would turn up millions of matches. How could you possibly write software to compare those documents? The data in those documents isn't stored in any way that's directly comparable.

That's where infosets come in, because the idea is to regularize how data is stored in an XML document that, ultimately, is designed to let you work with thousands of such documents. The idea behind infosets is to set up an abstract way of looking at an XML document that allows it to be compared to others. (Note that documents need to be well-formed to have an infoset.)

An XML infoset can contain fifteen different types of information items:

- A document information item
- Element information items
- Attribute information items
- Processing instruction information items
- Reference to skipped entity information items
- Character information items
- Comment information items
- A document type declaration information item
- Entity information items

- Notation information items
- Entity start marker information items
- Entity end marker information items
- CDATA start marker information items
- CDATA end marker information items
- Namespace declaration information items

So what software works with infosets? None, really—infosets are primarily theoretical constructs, and the infoset specification is mostly designed to provide a set of definitions that other XML specifications can use when they need to refer to the information in an XML document. Although the term *infoset* has entered common usage as a way to refer to the information in an XML document, it's not a specific enough specification to allow any real implementation. The closest you can come these days to truly regularizing the data in XML documents to make it easy to compare them is to use canonical XML, coming up next.

3

Understanding Canonical XML

Infosets are only abstract formulations of the information in an XML document. So without reducing an XML document to its infoset, how can you actually approach the goal of being able to actually compare XML documents character by character? You can write your documents in canonical XML.

TIP

You can find a canonical XML tutorial at www.xfront.com/canonical/CanonicalXML.html.

Canonical XML is a companion specification to XML, and you can read all about it at <http://www.w3.org/TR/xml-c14n>. Canonical XML is a very strict XML syntax, which lets documents in canonical XML be compared directly.

Using this strict syntax makes it easier to see whether two XML documents are the same. For example, a section of text in one document might read `Black & White`, whereas the same section of text might read `Black & White` in another document, and even `<![CDATA[Black & White]]>` in another. If you compare those three documents byte by byte, they'll be different. But if you write them all in canonical XML, which specifies every aspect of the syntax you can use, these three documents would all have the same version of this text (which would be `Black & White`) and could be compared without problem.

As you might imagine, the canonical XML syntax is very strict; for example, canonical XML uses UTF-8 character encoding only, carriage-return linefeed pairs are replaced with linefeeds (that is, `
`); tabs in CDATA sections are replaced by spaces, all entity references must be expanded, and much more, as specified in <http://www.w3.org/TR/xml-c14n>.

TIP

In their canonical form, documents can be compared directly, and any differences will be readily apparent. Because canonical XML is intended to be byte-by-byte correct, it's often a good idea to use software to convert your XML documents to that form. One such package that will convert valid XML documents to canonical form comes with the XML for Java software that you can get free from IBM's AlphaWorks (<http://www.alphaworks.ibm.com/tech/xml4j>). The actual program is named DOMWriter, and it's part of the XML for Java package.

That completes today's discussion on constructing XML documents. We've covered everything we need to know before we start discussing how to create valid XML documents—and we're going to start doing that tomorrow.

Summary

Today, you took a look at how to create well-formed XML documents. W3C doesn't even consider an XML document to be XML unless it's well-formed. W3C considers an XML document well-formed if it meets three criteria:

- Taken as a whole, it matches the production labeled *document*.
- It meets all the well-formedness constraints given in this specification (that is, the XML 1.0 specification, <http://www.w3.org/TR/REC-xml>).
- Each of the parsed entities, which is referenced directly or indirectly within the document, is well-formed.

The most general of these items says that an XML document must meet the well-formedness constraints in the XML specification, and you took a look today at what that meant.

Those constraints include beginning a document with an XML declaration, using only legal character references, the document must include at least one element, elements must be structured and nested correctly, the root element must contain all other elements, attribute names must be unique, attribute values must be quoted, and so on.

You also took a look at creating namespaces, and how namespaces help you avoid conflicts in XML. To define a namespace, you can assign the `xmlns:prefix` attribute to a unique identifier (usually a URI), or you can use the `xmlns` attribute to define a default namespace.

Q&A

Q Can I use an XML validator to test an XML document's well-formedness?

A Yes, if you have a DTD or XML schema for the document—an XML validator will also report whether the document is well-formed or not. However, you do need a DTD or XML schema if you want to use a validator—very few will check a document without one. One program that will check an XML document's well-formedness without a DTD or XML schema is Internet Explorer. If the document is not well-formed, you'll see the message "The XML page cannot be displayed", and Internet Explorer will tell you the exact problem with the document.

Q Do I need to use namespaces if there's no chance of tag name conflicts with other XML applications?

A Often, yes. Namespaces aren't used solely to avoid tag (and attribute) name conflicts—using a namespace also indicates to an XML processor what XML application you're using. For example, if you're using MathML, you must use the current MathML namespace or most MathML-enabled XML processors will complain.

Workshop

This workshop tests whether you understand the concepts you saw today. It's a good idea to make sure you can answer these questions before pressing on to tomorrow's work.

Quiz

1. To be well-formed, what's the least number of elements an XML document can contain?
2. Why is the following XML document not well-formed?

```
<?xml version = "1.0" standalone="yes"?>
<employee>
  <name>Frank</name>
  <position>Chef</position>
</employee>
<employee>
  <name>Ronnie</name>
  <position>Chef</position>
</employee>
```

3. Why is the following XML document not well-formed?

```
<?xml version = "1.0" standalone="yes"?>
<employee>
  <kitchen_staff/>
  <name language=en>Frank</name>
  <new_hire />
  <position language=en>Chef</position>
</employee>
```

4. How can you create a namespace named `service` whose URI is `http://www.superduperbigco.com/customer_service`?
5. How could you set the default namespace in a set of XML elements to the URI `http://www.superduperbigco.com/customer_returns`?

INDEX

Symbols

& (ampersand), 161
& character (well-formed documents), 90-92
< character (well-formed documents), 89-92
: (colon), 133
, (comma), 116
{ } curly braces, 252
<!-- --> element, 415
- (minus sign), 17
[] operator, 503
% (percent sign), 23, 161
+ (plus sign), 17, 119-124
? (question mark), 119-124
“” (quotation marks), attributes, 89
; (semicolon), 161, 263
@* (wildcard character), 305
* (asterisk), 115, 118-124
* (wildcard character), 302

A

<a> element, 419-422, 474

abbreviated syntax, RDF, 679-680

abbreviations, XPath, 317-321

absolute location paths, XPath, 309

Abstract Windowing Toolkit (AWT), 589-595

accessing

data fields, DSOs, 709-711

document data, 47-48

AchieveForms, 507

Active Server Pages (ASP), 728-731

ActiveX Data Object (ADO.NET), 33

actor attribute, 649

Add Dataset dialog box, 793

Add Existing Item command (Project menu), 181

Add Existing Item dialog box, 181

Add Item command (Project menu), 766

Add New Item dialog box, 766

Add Table dialog box, 733

Add Web Reference dialog box, 793

ADO.NET (ActiveX Data Object), 33

Adobe Web site, 340

Adobe FrameMaker, 44

alignment styles, CSS, 269-270

alink attribute, 394

all groups, declaring, 222

all schema element, 193

Amaya Web browser, 27, 488-489

American Standard Code for Information Interchange (ASCII), 52

ampersand (&), 161

ancestor axis, 310

ancestor-or-self axis, 310

<animate> element, 472

animations (SVG), creating, 472-474

annotations

DTDs, 127-128

schemas, 192-193, 233-234

text formatting, XHTML, 415

writing, 60-61

anonymous types, definitions, 215-217

any schema element, 193

anyAttribute schema element, 193

Apache XML Project's FOP, downloading, 339

appinfo schema element, 193

applets, DSO (data source object), 694

data binding HTML elements to XML data, 711-714

data binding HTML tables to XML data, 714-716

applications

storing as XML, 731-732

Web, creating, 787

XML, 26-27

arcs (extended XLinks), creating, 495-497

ASCII (American Standard Code for Information Interchange), 52

ASP (Active Server Pages), 728-731

asterisk (*), 115, 118-124

<!ATTLIST> element, 142

attachments, adding (SOAP messages), 659

Attr interface methods, 578

attribute axis, 310

attribute schema element, 193

attributeFormDefault attribute, 224

attributeGroup schema element, 193

attributes

<a> element, 420-421

actor, 649

alink, 394

anonymous type definitions, schemas, 216

assigning values, 69-70

attributeFormDefault, 224

 element, 408-409

background, 394

bgcolor, 394

 element, 400

<center> element, 401

- <circle> attribute, 460
- class, 259, 394
- colspan, 440
- d, 467
- default, 201
- defined, 142
- dir, 394
- <div> element, 402-403
- doctype-public, 328
- doctype-system, 328
- documents, reading (JavaScript), 551-553
- Dublin Core language, 674
- element attributes, viewing in Internet Explorer, 67
- elementFormDefault, 224
- elements, SAX, 616-618
- <ellipse> element, 461
- empty elements, schemas, 217
- encoding attribute (declarations), 59
- encoding, 84, 327
- encodingStyle, 649
- event, 476
- fill, 467
- fixed, 201
- element, 412
- form, 230
- <frame> element, 429-430
- <frameset> element, 428
- freeze, 473
- <h1> to <h6> elements, 407
- handling
 - node matching, 302-305
 - reading documents, 577-578
- <head> element, 392-393
- height, 456
- <html> element, 391
- <i> element, 410
- id, 394, 466
- ID, handling (node matching), 306
- element, 425-426
- indent, 327
- lang, 394
- <link> attribute, 422-423
- link, 394
- looping, 556, 617-618
- match (node matching), 301-308
- maxOccurs, 201
- minOccurs, 201
- mustUnderstand, 650
- naming, 68
- omit-xml-declaration, 328
- <p> element, 398-399
- <polyline> element, 462
- <rdf:Description> element, 672
- rel, 423-424
- resources, RDF, 677
- schemas, 198-203, 221-222
- select, XPath, 309
 - abbreviations, 317-321
 - axes, 310-311
 - default rules, 317-321
 - node tests, 311
 - predicates, 311-316
 - tools, 321
- SOAP (Simple Object Access Protocol), 649-650
- element, 405
- standalone, 60, 80, 328, 387
- <style> element, 433
- style, 255, 395, 434-435
- SVG, 459
- <table> element, 435-437
- targetNamespace, 223
- <td> element, 440-442
- text, 395
- <th> element, 438-440
- <title> element, 393-394
- title, 395
- <tr> element, 437-438
- transform, 471

- <u> element, 411
- use, 202-203
- value, 203
- version attribute (declarations), 59
- version, 84, 328
- vlink, 395
- well-formed documents, 88-92
- width, 456
- writing, 65-68
- XML declarations, 15
- xml:lang, 69-70, 395
- xml:space, 57
- xmlns, 98
- xmlns:prefix, 93, 97

attributes (DTDs)

- declaring, 142-144
 - default values, 145-150
 - type values, 145-146, 150-160
- nonwhitespace, 152
- whitespace, 153

Attributes object, 616-617

AWT (Abstract Windowing Toolkit), 589-595

axes, 310-311

B

** element, 408-410**

background attribute, 394

background styles, CSS, 265-268

background-attachment property, 265, 271

background-color property, 265

background-image property, 266, 271

background-position property, 271

background-repeat property, 266, 271

barenames (XPointers), 499

bgcolor attribute, 394

binding. *See* data binding

block content, text (XHTML), 402-405

block-level element, 253, 264

<body> element, 647-648, 394-397

body, SOAP (Simple Object Access Protocol) messages, 647

bold element, text (XHTML), 408-410

boolean expressions (XPath), 312

Booleans, select (XForms), 515-516

border styles, CSS, 268-269

border-bottom-width property, 268

border-color property, 268

border-left-width property, 268

border-right-width property, 268

border-style property, 268

border-top-width property, 268

border-width property, 268

bottom property, 275

boxes, multiline text, 743. *See also* dialog boxes

browsers. *See* Web browsers

**
 element, 400**

buttons

- Get Names, 744
- Get the Data, 796
- Query Builder, 733
- Read XML Data, 731
- Store XML Data, 731
- XForms
 - creating, 513-514
 - Reset, 516-517
 - Submit, 516-517

C

C#, csc command-line compiler, 650

callback methods, SAX, 613

candidate recommendations (W3C file), 13

canonical XML, 103-104

Capitalizer class, 651

- carriage returns, 58
- Cascading Style Sheets. *See* CSS
- CDATA sections, 70-73, 580
- CDATA type value, DTD attributes, 150-151
- ceiling() function, 315
- <center> element, 400-402
- centering text, XHTML, 400-402
- changeHandler method, 562
- character data. *See* CDATA type value
- character encodings, 52-54
- character entity references, 55-56
- character references, 54, 85
- character-points (XPointer schemes), 502
- characters
 - & (well-formed documents), 90-92
 - < (well-formed documents), 89-92
 - sensitive, style sheets, 434
 - wildcard (*), 302
 - wildcard (@*), 305
- characters method, 618
- Chemical Markup Language (CML), 28-29
- Chiba, 507
- child axis, 310
- child elements, 84
 - element content models, DTDs, 114-116
 - mixed content models (DTDs), 124
 - mixed-content elements, schemas, 219
 - multiple, element content models (DTDs), 118-124
- child nodes, 502, 579
- childLoop method, 554-557, 573-576
- <circle> element, 460
- choice schema element, 193
- choices
 - DTDs, 122-124
 - schemas, creating, 214
 - syntax (sample document), 122-123
- circles (SVG), creating, 460
- class attribute, 259, 394
- classes
 - Capitalizer, 651
 - DataGrid, 738-740
 - DataSet, 736
 - DocumentBuilder, methods, 571
 - SAXParser, methods, 612-613
 - SoapFormatter, 651
 - style, creating, 257-260
 - XmlTextWriter, 777
- client-side XSLT, 290-291
- clients, SOAP, 656
 - creating, 653-655, 662-668
 - installing, Tomcat servers, 665-666
- closing tags, elements, 15
 - reading documents, 580-584
 - SAX, 619
- CML (Chemical Markup Language), 28-29
- code. *See* syntax
- collapsed ranges, 503
- colon (:), 133
- color property, 266
- color styles, CSS, 265-268
- colors
 - predefined, SVG, 453-454
 - SVG, hexadecimal numbers, 454
- colspan attribute, 440
- columns
 - headers, spanning, 440
 - XHTML documents, 428-429
- command-line compilers, csc (C#), 650
- command-line prompts, % (percent sign), 23
- commands
 - d attribute, 468
 - Data menu
 - Generate Dataset, 791
 - Generate Dataset in Visual Basic .NET, 736
 - File menu, New, Project, 788-790

- Project menu
 - Add Existing Item, 181
 - Add Item, 766
 - Set as StartUp Project, 792
- Tools menu, Connect to Database, 732
- comma (,), 116**
- comment() node test, 311**
- comments. *See* annotations**
- comparing documents. *See* canonical XML**
- compilers, command-line, 650**
- complex schema element type, 195, 198-200**
- complex types, schemas, 217-218, 770-771**
- complexContent schema element, 193**
- complexType schema element, 193**
- concat(string1, string2,) function, 316**
- Connect to Database command (Tools menu), 732**
- connection object, 658**
- connections**
 - CSS, 254-256
 - data sources, 732
 - schemas (Visual Basic .NET), 773-774
- constraints, well-formedness, 80**
 - attributes, 88-89
 - declarations, 84
 - element structure, 85-86
 - entity references, 89-92
 - legal character references, 85
 - nesting elements, 87-88
 - root elements, 85-87
- contains(string1, string2) function, 316**
- content handling (DTDs)**
 - element content models, 114
 - text, element content models, 116-118
- content models. *See* element content models**
- context code, XQuery, 753-757**
- controls**
 - input, 509, 512
 - MSHTML (Microsoft HTML), 694-696
 - output, 509
 - properties, 787
 - range, 509
 - secret, 509
 - select, 509, 512-513
 - select1, 509
 - submit, 509
 - TDC (tabular data control), 694
 - textarea, 509
 - trigger, 509
 - upload, 509
 - Visual Studio .NET, formatted XML, 784-789
 - XForms, 508-509
- count function (XQuery), 758-759**
- count(node-set) function, 313**
- csc command-line compiler (C#), 650**
- CSS (Cascading Style Sheets), 18, 249-251**
 - documents, connecting, 254-256
 - inline styles, 262-263
 - properties, 253, 265
 - resources, 252
 - style rule specification
 - alignment styles, 269-270
 - background styles, 265-268
 - block-level elements, 264
 - border styles, 268-269
 - color styles, 265-268
 - image styles, 270-274
 - list styles, 278
 - margin styles, 269-270
 - positioning styles, 274-277
 - semicolon (;), 263
 - table styles, 279-281
 - text styles, 264-265
- style sheet selectors**
 - creating, 256-257
 - grouping, 256
 - ID values, 261-262
 - style classes, 257-260
- syntax, 19**
- validators, 253**

CSS styles, SVG documents, 457-459
 CSS1, 252
 CSS2, 252
 CSS3, 252
 curly braces { }, 252
 current records, DSOs (data source objects), 696

D

d attribute, 467

data

- displaying via style sheets, 18
- extracting from documents (JavaScript), 554-560
- extracting from files, 20-23
- handling (Visual Basic .NET), 776
- non-XML associating with XML documents, 168-171
- packaging, 13
- parsed character, DTDs, 116
- separating from presentations (XForms), 510-512
- structure, 24
- tables (XHTML), 440-443
- text, mixed content models (DTDs), 124
- validating, 775
- XML, storing in RDF, 678-679

Data Adapter Configuration Wizard, 732, 744, 791

data binding

- DSOs (data source objects), 694-696, 709-711
- HTML elements to HTML data, 696-703
- HTML elements to XML data, 703-706, 711-714

- HTML tables to XML data, 706-709, 714-716
- Internet Explorer, 694
- XML data
 - hierarchical, 720-725
 - searches, DSOs/JavaScript, 716-719

Data Connections icon, 732

data fields (DSOs), accessing, 709-711

data islands, data binding, 547-549, 694

- HTML elements to XML data, 704-705
- HTML tables to XML data, 706

Data Link Properties dialog box, 732

Data menu commands

- Generate Dataset, 791
- Generate Dataset in Visual Basic .NET, 736

data objects, 78

data source objects (DSOs)

- data binding, 694-696, 717-719
 - HTML elements to XML data, 711-714
 - HTML tables to XML data, 714-716
- data fields, accessing, 709-711
- events, 703
- HTML documents, 700-701
- record sets, 696, 701-703
- records, 696, 699-701

data sources, connections, 732

data types (schemas), specifying, 208

databases

- ASP (Active Server Pages), 728-731
- storing as XML, 731-743
- XPath, 743-749
- XQuery
 - implementations, 750-752
 - Lucent Galax XQuery processor, 752-761
 - online working drafts, 750
 - W3C, 749-750

DataGrid class, 738-740

DataSet class, 736

Dataset Properties dialog box, 776

datasets, creating, 736

declarations

attributes

- assigning values, 69-70
- encoding attribute, 59
- naming, 68
- standalone attribute, 60
- version attribute, 59
- writing, 65-68

global, schema elements, 223

local, schema elements, 223-232

schemas, 192

well-formed documents, 84

writing, 59-60

XForms, 509

XHTML documents, 387

declaring

all groups, 222

class attribute, 259

DTD attributes, 142-144

default values, 145-150

type values, 145-146, 150-160

DTD entities, 162-165

empty elements, schemas, 217-218

mixed-content elements, schemas, 218-219

schemas

attributes, 198-200

elements, 195, 198-200

simple scheme types, 197

unparsed external DTD entities, 168

default attribute, 201**default namespaces, creating, 98-102****default values**

DTD attributes, 145-146

#FIXED (DTD attributes), 149-150

immediate (DTD attributes), 146-147

#IMPLIED (DTD attributes), 148-149

#REQUIRED (DTD attributes), 147-148

schema elements, 201

DefaultHandler object, 611-614

definitions, anonymous types (schemas), 215-217

descendant axis, 310

descendant-or-self axis, 310

descendants elements, 302

descriptions, RDF (Resource Description Framework), 672, 676-678

dialog boxes

Add Dataset, 793

Add Existing Item, 181

Add New Item, 766

Add Table, 733

Add Web Reference, 793

Data Link Properties, 732

Dataset Properties, 776

Generate Dataset, 736, 744

New Project, 788-790

dir attribute, 394

directed label graphs. *See* extended XLinks

directives, 174-176

display property, 264

<div> element, 402-405

<!DOCTYPE> element, 109-110, 131, 387-389

doctype-public attribute, 328

doctype-system attribute, 328

<document> element, * (asterisk), 115

document elements, 16, 772-773

document nodes, 576-577

Document object, 572-573

Document Object Model. *See* DOM

document production, well-formed documents, 79

document type definitions. *See* DTDs

documentation schema element, 193

DocumentBuilder object, methods, 571

DocumentBuilderFactory object, 569-570

documents. *See also* well-formed documents;

XSLT, style sheets

accessing data, 47-48

attributes, looping (JavaScript), 556

- comparing. *See* canonical XML
- connecting to schemas (Visual Basic .NET), 773-774
- creating (Visual Basic .NET), 766-767
- CSS, connecting, 254-256
- data
 - adding (Visual Basic .NET), 774-775
 - extracting, 20-23, 554-560
- data structure, 24
- editing, 601-604
- element content models
 - child elements, 114-124
 - content handling, 114
 - creating, 113-114
 - empty elements, 126-127
 - mixed, 124-126
 - text content handling, 116-118
- elements, finding by name, 584-588, 623-628
- embedding, 729
- finding by name, 549-551
- frame (XHTML), creating, 427-429
- HTML, data holding, 694-695
- linking (XHTML), 422-425
- navigating, 596-597, 633-637
- nodes, looping, 554-557, 781
- parsing, 570
- PDF, 339-340
- reading with Java, 568-572
 - attributes, 577-578
 - CDATA sections, 580
 - child nodes, 579
 - childLoop method, 573-576
 - document nodes, 576-577
 - elements, 577, 580-584
 - processing instructions, 580
 - text nodes, 579
- reading with JavaScript, 544-546
 - attributes, 551-553
 - data islands, 547-549
- saving, 44
- schemas, validators, 49
- text editors, 44-46
- validators, 48-49
- sample (ch08_01.xml), 250-252
- sample (ch09_01.xml), syntax, 287
- sample (ch09_02.xml), syntax, 288
- schemas, creating (Visual Basic .NET), 767
- SMIL, creating, 479-482
- starting with SAX, 614
- SVG (Scalable Vector Graphics)
 - animations, creating, 472-474
 - circles, creating, 460
 - creating, 454-456
 - CSS styles, 457-459
 - ellipses, creating, 461
 - gradients, creating, 466-467
 - groups, creating, 471-472
 - hyperlinks, creating, 474-475
 - JavaScript, creating, 476-478
 - lines, creating, 462
 - paths, creating, 467-468
 - polygons, creating, 463-464
 - polylines, creating, 462-463
 - rectangles, creating, 456
 - text paths, creating, 469-470
 - text, creating, 464-465
 - titles, 455-456
 - transformations, creating, 471-472
 - validating, 455
 - viewing (SVG Viewer), 456-457
- transforming XSLT, 286-292
- trees, nodes, 293
- valid. *See* DTDs (document type definitions); schemas
- viewing, 17-20
- well-formed, 24-25, 78-79
- writing, 51
 - attributes, 65-70
 - CDATA sections, 70-73
 - character encodings, 52-54

- comments, 60-61
- declarations, 59-60
- elements, naming tags, 63
- empty elements, 64
- entities, 55-56, 73-74
- line endings, 58
- markups, 55-56
- processing instructions, 62
- prologs, 58-59
- root elements, 64
- whitespace, 57
- XForms, 504-505
 - buttons, 513-517
 - controls, 508-509, 512-513
 - declarations, 509
 - presentations, separating data from, 510-512
 - select Booleans, creating, 515-516
 - software, 506-507
 - writing, 509-510
- XHTML
 - columns, 428-429
 - declarations, 387
 - <!DOCTYPE> element, 387-389
 - requirements, 386
 - standalone attributes, 387
 - validating, 390-391
 - writing, 386-390
- XHTML/MathML, 100-101
- XPointers
 - barenames, 499
 - element schemes, 499-500
 - framework specifications, 499
 - namespace schemes, 500
 - software support, 498
 - XPointer, 500-504
- XSL-FP, creating with XSLT, 335-339
- doGet method, 658**

DOM (Document Object Model), 451

Java

- browsers, creating, 589-595
- document elements, finding by name, 584-588
- documents, 596-604
 - reading documents, 568-574
 - reading documents, attribute handling, 577-578
 - reading documents, CDATA sections, 580
 - reading documents, child nodes, 579
 - reading documents, childLoop method, 573-576
 - reading documents, document nodes, 576-577
 - reading documents, element closing tags, 580-584
 - reading documents, element handling, 577
 - reading documents, processing instructions, 580
 - reading documents, text nodes, 579

JavaScript, 534

- documents, extracting data, 554-560
- documents, looping attributes, 556
- documents, looping nodes, 554-557
- documents, validating with DTDs, 560-564
- reading documents, 544-553

levels, 533-534

nodes, 532-533

objects, 534

- DOMDocument, 536-539
- programming, 535
- XMLDOMAttribute, 541-542
- XMLDOMELEMENT, 540-541
- XMLDOMNode, 539-540
- XMLDOMText, 542-544

overview, 532-533

DOMDocument object

- creating, 536
- methods, 538-539
- properties, 536-538

downloading

- Apache XML Project's FOP, 339
- Java, 568
- Lucent Galax XQuery processor, 752
- Saxon XSLT processor, 286

drivers, DTDs, 175**DSOs (data source objects)**

- data binding, 694-696, 716-719
- data fields, accessing, 709-711
- events, 703
- HTML documents, 700-701
- record sets, 696, 701-703
- records, 696, 699-701

DTD attributes

- declaring, 142-144
 - default values, 145-150
 - type values, 145-146, 150-160
- nonwhitespace, 152
- whitespace, 153

DTD entities, 160

- declaring, 162-165
- directives, 174-176
- external, 161, 168
- general, 161
- internal, 161
- non-XML data, associating with XML documents, 168-171
- parameter
 - % (percent sign), 161
 - ;(semicolon), 161
 - external, 172-174
 - internal, 171-172
- references
 - general, 161-168
 - nesting, 165

DTDs (document type definitions), 107

- * (asterisk), 118-119
- ? (question mark), 119
- choices, 122-124
- comments, 127-128
- documents, validating, 112-113, 560-564
- drivers, 175
- element content models
 - child elements, 114-124
 - content handling, 114
 - creating, 113-114
 - empty elements, 126-127
 - mixed, 124-126
 - text content handling, 116-118
- external, 128-133
- external subsets, 171
- internal, 131-133
- namespaces, 94, 133-135
- non-markup text (parsed character data), 116
- overview, 108-111
- parameter entities, 222
- parameterized, 175
- SVG (Scalable Vector Graphics), 455
- syntax, 25, 110-111
- Web browsers, 113

Dublin Core language (RDF), 670

- attributes, 674
- elements, 673-674
- resource types, 674-675
- storing XML data, 678-679

E**<!ELEMENT> element, 113****element content models**

- documents
 - child elements, 114-124
 - content handling, 114

- creating, 113-114
- empty elements, 126-127
- mixed, 124-126
- text content handling, 116-118
- mixed, 117
 - syntax (sample document), 124-125
 - XHTML, 397
- sequence, 116, 119-122
- subsequences, creating, 120
- element schema element, 194**
- element schemes (XPointer), 499-500**
- elementFormDefault attribute, 224**
- elements**
 - `<a>`, 419-422, 474
 - `<animate>`, 472
 - `<!ATTLIST>`, 142
 - attributes
 - assigning values, 69-70
 - naming, 68
 - SAX, 616-618
 - viewing in Internet Explorer, 67
 - writing, 65-68
 - `<b>`, 408
 - block-level, 253, 264
 - `<Body>`, 647-648
 - `<body>` (document body), 394-397
 - `<br>`, 400
 - `<center>`, 400-402
 - child, 84
 - element content models (DTDs), 114-116
 - mixed-content models (DTDs), 124
 - mixed-content elements (schemas), 219
 - multiple (element content models), 118-124
 - `<circle>`, 460
 - closing tags
 - reading documents, 580-584
 - SAX, 619
 - complex, 195, 198-200
 - creating, 15-17, 771-773
 - descendants, 302
 - description, RDF (Resource Description Framework), 672
 - `<div>`, 402-405
 - `<!DOCTYPE>`, 109-110, 131, 387-389
 - `<document>`, * (asterisk), 115
 - documents, 16, 549-551
 - Dublin Core language, 673-674
 - `<ellipse>`, 461
 - `<!ELEMENT>`, 113
 - `<EMBED>`, 478
 - empty elements, 64, 86, 126-127, 217-218
 - `<Envelope>`, 647
 - `<Fault>`, 647-649
 - finding by name, 584-588, 623-628
 - `<fo:block>`, 333, 350-353
 - `<fo:external-graphic>`, 357-359
 - `<fo:flow>`, 349-350
 - `<fo:inline>`, 354-357
 - `<fo:layout-master-set>`, 345
 - `<fo:list-block>`, 362
 - `<fo:list-item>`, 362
 - `<fo:list-item-body>`, 362
 - `<fo:list-item-label>`, 362
 - `<fo:page-number>`, 360-362
 - `<fo:page-sequence>`, 348-349
 - `<fo:region-after>`, 347-348
 - `<fo:region-body>`, 347-348
 - `<fo:root>`, 344-345
 - `<fo:simple-page-master>`, 345-347
 - `<fo:static-content>`, 349
 - `<fo:table>`, 365
 - `<fo:table-body>`, 365
 - `<fo:table-cell>`, 365
 - `<fo:table-column>`, 365
 - `<fo:table-headers>`, 365
 - `<fo:table-rows>`, 365
 - `<frame>`, 429-432
 - `<frameset>`, 427-429
 - `<friends>`, 720

- <g>, 471
- global declarations, schemas, 223
- grandchildren, 84
- <h1> to <h6>, 406-408
- handling
 - node matching, 301-302
 - reading documents, 577
- <head> (document head), 392-393
- <Header>, 647
- <html> (document element), 391-392
- HTML
 - binding to HTML data, 696-703
 - binding to XML data, 703-706, 711-714
 - Internet Explorer, 697-698
- , 425-427
- <item>, 513
- <label>, 514-515
- <line>, 462
- <linearGradient>, 466
- <link>, 422-425
- local declarations, schemas, 223-232
- <message>, 514
- mixed-content, schemas, 218-219
- <name>, syntax, 299
- naming, 15
- nesting (well-formed documents), 87-88
- <p>, 398-399
- <par>, 480
- <paragraph>, syntax, 272
- parents, 84
- <path>, 467-469
- <polygon>, 463
- property, RDF (Resource Description Framework), 672
- <rdf:Description>, 672
- <rect>, 456
- relationships, 84
- <reset>, 516
- root elements, 16, 64
 - adding (well-formed documents), 80-81
 - RDF (Resource Description Framework), 671
 - well-formed documents, 79, 85-87
- schemas, 193-194
 - declaring, 195, 198-200
 - default values, 201
 - grouping, 219-220
 - number specification, 200-201
 - type, 195
- <select>, 512
- <select1>, 512
- <selectboolean>, 515
- <seq>, 480
- simple, 195-197
- siblings, 83
- SOAP (Simple Object Access Protocol), 647-649
- , 405-406
- starting with SAX, 615-616
- structure (well-formed documents), 85-86
- <style>, 432-435
- <submit>, 516
- SVG, 452
- <table>, 435-437, 706
- tags
 - naming, 63
 - opening/closing, 15
- <td>, 440-443
- <text>, 464
- <textpath>, 469-470
- <th>, 438-440
- <title> (document title), 393-394
- <tr>, 437-438
- <trigger>, 514
- <XML>, 548
- <xsd:annotation>, 233
- <xsd:appInfo>, 233-234

- `<xsd:documentation>`, 233-234
- `<xsd:element>`, 195
- `<xsl:apply-templates>`, 295-298
- `<xsl:choose>`, 324-328
- `<xsl:copy>`, 321-322
- `<xsl:for-each>`, 298-300
- `<xsl:if>`, 323-324
- `<xsl:output>`, 327
- `<xsl:value-of>`, 298-300
- elements**`<polyline>`, 462
- `<ellipse>` element, 461
- ellipses (SVG), creating, 461
- `<EMBED>` element, 478
- embedded style sheets, creating, 432-435
- embedding**
 - documents, 729
 - SVG in HTML, 478
- empty elements**, 64, 86
 - content models (DTDs), 126-127
 - schemas, declaring, 217-218
 - syntax (DTDs), 126-127
- emulating hyperlinks, XLinks**, 488-489
- encoding attribute**, 15, 59, 84, 327
- encodingStyle attribute**, 649
- entity references**, 55-56, 73
- entities**, 55-56
 - defined, 142, 160
 - parameter, DTDs, 222
 - parsed, well-formed, 80
 - parsing, 73
 - unparsed entities, 74
- entities (DTDs)**, 160
 - declaring, 162-165
 - directives, 174-176
 - external, 161, 168
 - general, 161
 - internal, 161
 - non-XML data, associating with XML documents, 168-171
 - parameter, 161, 171-174
 - references, 161-168
- ENTITIES type value, DTD attributes**, 157-158
- entity references, well-formed documents**, 89-92
- ENTITY type value, DTD attributes**, 156-157
- enumerated type value, DTD attributes**, 151-152
- enumeration schema facet**, 210
- `<Envelope>` element, 647
- envelopes, SOAP (Simple Object Access Protocol) messages, 647
- error method**, 619
- errors**
 - nesting, 24-25
 - SAX, handling, 619-623
- event attributes**, 476
- events, DSOs (data source objects)**, 703
- Explorer (Internet)**
 - data binding, 694
 - HTML elements, 697-698
- expressions**
 - boolean (XPath), 312
 - regular, 209
 - XPath, 308-309
- extended XLinks**
 - arcs, creating, 495-497
 - inline links, 494
 - linkbases, creating, 497-498
 - out-of-line links, 494, 497-498
- extending XHTML**, 443-446
- Extensible Hypertext Markup Language. *See* XHTML**
- Extensible Stylesheet Language Transformation. *See* XSLT**
- Extensible Stylesheet Language. *See* XSL**
- extension schema element**, 194
- extensions. *See* file extensions**

external DTDs, 132-133

- entities, 161, 168-171
- private, creating, 128-130
- public, creating, 130-131

external general entity references (DTDs), creating, 165-168**external parameter DTD entities, creating, 172-174****external style sheets, syntax, 425****external subsets, DTDs, 171****F****facets (schemas)**

- simple ordered schema types, 212-213
- simple schema types, 208-213

fatalError method, 619**<Fault> element, 647-649****field schema element, 194****fields**

- data (DSOs), accessing, 709-711
- org.w3c.dom.Node object, 574-575

file extensions

- saving documents, 44
- .svg, 455
- WordPad issues, 44

File menu commands, New, Project, 788-790**files. *See* documents****fill attribute, 467****fixed attribute, 201****#FIXED default values, DTD attributes, 149-150****floor() function, 315****flow, page sequences, 349-350****following axis, 310****following-sibling axis, 310****<fo:block> element, 333, 350-353****<fo:external-graphic> element, 357-359****<fo:flow> element, 349-350****<fo:inline> element, 354-357****<fo:layout-master-set> element, 345****<fo:list-block> element, 362****<fo:list-item> element, 362****<fo:list-item-body> element, 362****<fo:list-item-label> element, 362**** element, 411-414****font styles, 265****font-family property, 264****font-size property, 264****font-style property, 264****font-weight property, 264****FOP (Apache XML Project), downloading, 339****<fo:page-number> element, 360-362****<fo:page-sequence> element, 348-349****<fo:region-after> element, 347-348****<fo:region-body> element, 347-348****foreground images, 273-276****form attribute, 230****formal public identifier (FPI), 130****format-number(number1, string2, string3) function, 316****formatted XML, displaying (Visual Basic .NET controls), 784-789****formatting. *See also* CSS (Cascading Style Sheets); XSL-FO (XSL Formatting Objects); XSLT (Extensible Stylesheet Language Transformation)****inline (XSL-FO), 353****<fo:external-graphic> element, 357-359****<fo:inline> element, 354-357****<fo:page-number> element, 360-362****lists (XSL-FO), 362-364****tables****XHTML, 435-437****XSL-FO, 365-368****text (XHTML), 408-415****FormFaces, 507**

forms, XForms, 504-505

- buttons, 513-517
- controls, 508-509, 512-513
- declarations, 509
- presentations, separating data from, 510-512
- select Booleans, creating, 515-516
- software, 506-507
- writing, 509-510

FormsPlayer, 506

- <fo:root> element, 344-345**
- <fo:simple-page-master> element, 345-347**
- <fo:static-content> element, 349**
- <fo:table> element, 365**
- <fo:table-body> element, 365**
- <fo:table-cell> element, 365**
- <fo:table-column> element, 365**
- <fo:table-header> element, 365**
- <fo:table-rows> element, 365**
- FPI (formal public identifier), 130**
- fractionDigits schema facet, 210**
- <frame> element, 429-432**
- frame documents (XHTML), creating, 427-429**
- frames (XHTML), creating, 429-432**
- <frameset> element, 427-429**
- frameset XHTML (Extensible Hypertext Markup Language), 30**
- Frameset XHTML 1.0 DTD, 385**
- freeze attribute, 473**
- <friends> element, 720**
- FrameMaker, 44**
- functions**
 - node sets, 313
 - numbers (XPath), 315
 - point(), 503
 - ranges (XPointer schemes), 503
 - strings (XPath), 316
 - XPath functions, XPointer schemes, 501
 - XQuery, 753-754, 758-759

G

- <g> element, 471**
- Galax XQuery processor (Lucent), 752-761**
- GEDCOM (Genealogical Data Communication), 26**
- Genealogical Data Communication (GEDCOM), 26**
- general DTD entities, 161**
- general entity references, 55-56, 73, 161**
 - external, 165-168
 - internal, 162-165
- Generate Dataset command (Data menu), 791**
- Generate Dataset dialog box, 736, 744**
- Generate Dataset in Visual Basic .NET command (Data menu), 736**
- Get Names button, 744**
- Get the Data button, 796**
- getAttributes method, 577**
- getNodeName method, 577**
- getNodeValue method, 577**
- global element declarations, 223**
- globally declaring schema elements, 200**
- gradients (SVG), creating, 466-467**
- grammar. *See* syntax**
- grandchildren elements, 84**
- graphic objects, SVG, 451**
- graphics. *See also* SVG (Scalable Vector Graphics)**
 - animations, creating, 472-474
 - foreground, positioning, 275-276
 - VML (Vector Markup Language), 450-451
 - XHTML documents, 425-427
- group schema element, 194**
- grouping**
 - attributes, schemas, 221-222
 - elements, schemas, 219-220
 - style sheet selectors, CSS, 256

groups

- all, declaring, 222
- SVG, creating, 471-472

H

<h1> to <h6> elements, 406-408

handling content, element content models (DTDs), 114-118

<head> element (document head), 392-393

<Header> element, 647

headers

- columns, spanning, 440
- SOAP (Simple Object Access Protocol) messages, 647
- tables (XHTML), 438-440

headings, text (XHTML), 406-408

height attribute, 456

hexadecimal numbers, colors (SVG), 454

hierarchical XML data, data binding, 720-725

HiT Software Web site, 181

<html> element (document element), 391-392

HTML (Hypertext Markup Language)

- data binding, 696-706
- documents
 - data holding, 694-695
 - verification, 185-188
- DSO document, 700-701
- elements
 - binding to XML data (DSO applet), 711-714
 - Internet Explorer, 697-698
- hyperlinks. *See* XLinks
- overview, 12
- sample Web page, 10
- SVG, embedding in, 478

tables

- binding to XML data (DSO applet), 714-716
- data binding to XML data, 706-709
- tags, 10-12

HTML+TIME, 32-33

hyperlinks. *See also* XLinks

- emulating, XLinks, 488-489
- SVG, creating, 474-475
- XHTML, creating, 419-422

Hypertext Markup Language. *See* HTML

I

<i> element, 410-411

icons

- Data Connections, 732
- Windows Application, 792

id attribute, 394, 466

ID attributes, match (node matching), 306

ID type value, DTD attributes, 154-155

ID values, style sheet selectors, 261-262

id(ID) function, 313

identifiers, FTP (formal public identifier), 130

IDREF type value, DTD attributes, 155-156

IGNORE DTD entity directive, 174-176

IIS (Internet Information Server), Web applications, 787

image styles, CSS, 270-274

images. *See* graphics

** element, 425-427**

immediate default values, DTD attributes, 146-147

implementing XLinks, 486-487

#IMPLIED default values, DTD attributes, 148-149

import schema element, 194

INCLUDE DTD entity directive, 174-176

- include schema element, 194**
- incomingMessage object, 660**
- indent attribute, 327**
- infosets (information sets), 102-103**
- inline content, text (XHTML), 405-406**
- inline formatting (XSL-FO), 353**
 - <fo:external-graphic> element, 357-359
 - <fo:inline> element, 354-357
 - <fo:page-number> element, 360-362
- inline links (extended XLinks), 494**
- inline styles**
 - creating, 255, 434
 - CSS, 262-263
- input controls, 509, 512**
- installations, SVG Viewer, 457**
- interface methods**
 - Attr, 578
 - Attribute object, 617
- interfaces, org.w3c.dom.Node, 575-576**
- internal DTDs, 131-133, 161**
- internal general entity references (DTDs), creating, 162-165**
- internal parameter DTD entities, creating, 171-172**
- internal style sheets. *See* embedded style sheets**
- Internet Explorer, 47, 184**
 - data binding, 694
 - HTML elements, 697-698
 - MSXML, 184-185
 - viewing CDATA sections, 72
 - viewing element attributes, 67
 - XLinks, 489
- Internet Information Server (IIS), Web applications, 787**
- ISO 10646. *See* UCS (Universal Character System)**
- italic element, text (XHTML), 410-411**
- <item> element, 513**

J

Java

- browsers, creating, 589-595
- documents
 - elements, finding by name, 584-588
 - navigating, 596-597
 - reading, 568-574
 - reading, attributes, 577-578
 - reading, CDATA sections, 580
 - reading, child nodes, 579
 - reading, childLoop method, 573-576
 - reading, document nodes, 576-577
 - reading, element closing tags, 580-584
 - reading, elements, 577
 - reading, processing instructions, 580
 - reading, text nodes, 579
- downloading, 568
- files, data, extracting, 22-23
- SOAP example, 657
 - clients, creating, 662-668
 - clients, installing, 665-666
 - servers, creating, 658-662
 - servlets, 656
- whitespace, 596
- XML, writing, 597-604
- XSLT, 291-292
- Java Web Services Developer's Pack, 657**
- Java XML Messaging (JAXM), 657**
- Java XML Pack, 657**
- Java/SAX**
 - browsers, creating, 628-633
 - XML, writing, 637-641
- JavaScript, 185, 534**
 - documents
 - attributes, looping, 556
 - data, extracting, 554-560
 - nodes, looping, 554-557
 - reading, 544-553
 - validating with DTDs, 560-564

- DSOs, data field access, 709-711
- files, data, extracting, 20-21
- SVG, creating, 476-478
- XML data searches, 716-719

JAXM (Java XML Messaging), 657

JScript. *See* JavaScript

JSP, server-side XSLT, 288-289

Jumbo, 28, 48

jXForms, 507

K-L

key schema element, 194

keyref schema element, 194

keywords

- PUBLIC, 130, 165
- SYSTEM, 128, 165

<label> element, 514-515

lang attribute, 394

languages. *See also* markup languages

- Dublin Core (RDF), 670
 - attributes, 674
 - elements, 673-674
 - resource types, 674-675
 - storing XML data, 678-679
- query. *See* XQuery
- SMIL (Synchronized Multimedia Integration Language), 29-30

last() function, 313

left property, 275

legal character references, well-formed documents, 85

length schema facet, 210

levels, DOM, 533-534

<line> element, 462

<linearGradient> element, 466

line breaks, text (XHTML), 400

line-height property, 264, 269

linefeed characters, 58

lines (SVG), 462-463

<link> element, 422-425

link attribute, 394

linkbases (extended XLinks), creating, 497-498

linking

- documents (XHTML), 422-425
- XBase, 504

links. *See* hyperlinks

linksets (extended XLinks), 497

LiquidOffice, 507

list formatting (XSL-FO), 362-364

list schema element, 194

list styles, CSS, 278

list-item property, 278

list-style-image property, 278

list-style-type property, 278

local element declarations, 223-232

local namespaces, creating, 97-98

local-name(node-set) function, 313

locally declaring schema elements, 200

location steps, XPath, 309, 317-321

locations (XPath), XPointer schemes, 501

logical operators, 312

looping

- attributes, 556, 617-618
- nodes, 554-557, 574-576, 781

loops, While, 781

Lucent Galax XQuery processor, 752-761

M

margin styles, CSS, 269-270

margin-bottom property, 269

margin-left property, 270

margin-right property, 270

margin-top property, 270**markup languages**

- CML (Chemical Markup Language), 28-29
- HTML, 12
- MathML (Mathematical Markup Language), 27-28
- overview, 10-12
- SGML (Standard Generalized Markup Language), 13
- XHTML (Extensible Hypertext Markup Language), 30-31

markups

- general entity references, 55-56, 73
- parameter entity references, 55-56

master set layouts, <fo:root> element, 345**master templates, creating, 345****masters, page masters, 345-347****match attribute (node matching)**

- attributes, handling, 302-305
- elements, handling, 301-302
- ID attributes, handling, 306
- multiple matching, handling, 306-308
- processing instructions, handling, 306
- root nodes, 301

matching XSLT, XPath expressions, 308-309**matching nodes (XSLT)**

- attributes, handling, 302-305
- elements, handling, 301-302
- ID attributes, handling, 306
- multiple matching, handling, 306-308
- processing instructions, handling, 306
- root nodes, 301

matching strings, ranges (XPointer schemes), 503**MathML (Mathematical Markup Language), 27-28****MathML/XHTML document, 100-101****maxExclusive schema facet, 210****maxInclusive schema facet, 209-210****maxLength schema facet, 210****maxOccurs attribute, 201****<message> element, 514****MessageFactory object, 658-659****methods**

- callback, SAX, 613
- changeHandler, 562
- characters, 618
- childLoop, 554-557, 573-576, 592, 598-599, 611
- DataGrid class, 740
- DataSet class, 736
- DefaultHandler object, 614
- Document object, 572-573
- DocumentBuilder object, 571
- DocumentBuilderFactory objects, 569-570
- doGet, 658
- DOMDocument object, 538-539
- DOMElement object, 541
- error, 619
- fatalError, 619
- getAttributes, 577
- getNodeName, 577
- getNodeValue, 577
- interface, 578, 617
- NamedNodeMap, 578
- NodeList, 579
- OleDbDataAdapter object, 735
- org.w3c.dom.Node interface, 575-576
- record sets, DSOs (data source objects), 702-703
- recursion, 554
- SAXParser class, 612-613
- SAXParserFactory object, 612
- ToUpper, 651
- upper, 651-654
- warning, 619
- WriteStartDocument, 778
- WriteStartElement, 778
- XmlDataDocument object, 746-747

- XMLDOMAttribute object, 542
- XMLDOMNode, 539-540
- XMLDOMText object, 543-544
- Microsoft .NET, 33-35**
- Microsoft HTML (MSHTML) control, 694-696**
- Microsoft Visual Studio .NET, 181-184**
- Microsoft Web site, 181, 184**
- MIME types, SVG (Scalable Vector Graphics), 455**
- minExclusive schema facet, 210**
- minInclusive schema facet, 209-210**
- minLength schema facet, 210**
- minOccurs attribute, 201**
- minus sign (-), 17**
- mixed content models, 117**
 - DTDs, 124-126
 - syntax (sample document), 124-125
 - XHTML, 397
- mixed-content elements, schemas, 218-219**
- Mosquito XForms, 506**
- MSHTML (Microsoft HTML) control, 694-696**
- MSXML, Internet Explorer, 184-185**
- multiline text boxes, creating, 743**
- multimedia sequences, creating, 480**
- multiple child elements, element content models (DTDs), 118-124**
- multiple matches, handling (node matching), 306-308**
- multiple resources, RDF, 675-677**
- mustUnderstand attribute, 650**

N

- <name> element, syntax, 299**
- name token. *See* NMTOKEN type value**
- name(node-set) function, 313**

- NamedNodeMap methods, 578**
- namespace axis, 310**
- namespace-uri(node-set) function, 313**
- namespaces**
 - colon (:), 133
 - creating, 92-93
 - default, creating, 98-102
 - defining with URIs, 93-97
 - DTDs, 133-135
 - Dublin Core language, 673
 - local, creating, 97-98
 - schemas, 192, 222-232
 - schemes (XPointer), 500
 - XSL-FO, 332
- naming**
 - attributes, 68
 - elements, 15
 - tags, 63
- navigation**
 - documents
 - Java, 596-597
 - SAX, 633-637
 - DSO records, 699-701
- nesting**
 - DTD entity references, 165
 - elements, well-formed documents, 87-88
 - errors, 24-25
 - RDF descriptions, 676-678
- .NET**
 - Microsoft, 33-35
 - SDK (Software Development Kit), 650
 - SOAP example, 650-656
 - Visual Basic, databases, storing as XML, 731-732
- Netscape Navigator, 47**
- New Project dialog box, 788-790**
- New, Project command (File menu), 788-790**
- newsgroups, Usenet, 39**
- NMatrix, 507**
- NMTOKEN type value, 152-153**

node matching (XSLT)

- attributes, handling, 302-305
- elements, handling, 301-302
- ID attributes, handling, 306
- multiple matches, handling, 306-308
- processing instructions, handling, 306
- root nodes, 301

node sets (XPath), 313-315**node tests, 308, 311****node() node test, 311****NodeList methods, 579****nodes**

- child, 502, 579
- document, 576-577
- DOM, 532-533
- looping, 554-557, 574-576, 781
- text, 579
- trees, 293
- XSLT, copying, 321

non-markup text, DTDs, 116**non-XML data, associating with XML documents, 168-171****nonwhitespace, DTD attributes, 152****normalize-space(string1) function, 316****notation schema element, 194****NOTATION type value, DTD attributes, 158-160****notes (W3C file), 13****Novell XForms, 507****numbers (XPath), 315-316****numeric operators, 315****O****objects**

- Attributes, 616-617
- connection, 658
- data, 78

DefaultHandler, 611-614

Document, methods, 572-573

DocumentBuilderFactory, 569-570

DOM, 534-536

DOMDocument, 536-539

DSOs (data source objects), data binding, 694-696

graphics, SVG, 451

incomingMessage, 660

MessageFactory, 658-659

OleDbDataAdapter, 735

org.w3c.dom.Node, fields, 574-575

SAXParserFactory, 611-612

SoapFormatter, 653

XmlDataDocument, 745-747

XMLDOMAttribute, 541-542

XMLDOMElement, 540-541

XMLDOMNode, 539-540

XMLDOMText, 542-544

XmlTextReader, 780

XSL-FO, 341-343

<fo:block> element, 350-353

<fo:flow> element, 349-350

<fo:layout-master-set> element, 345

<fo:page-sequence> element, 348-349

<fo:region-after> element, 347-348

<fo:region-body> element, 347-348

<fo:root> element, 344-345

<fo:simple-page-master> element, 345-347

<fo:static-content> element, 349

OleDbDataAdapter object, 735**omit-xml-declaration attribute, 328****online resources. *See* resources****opening tags (elements), 15****operators**

logical, 312

numeric, 315

[], 503

org.w3c.dom.Node interface, methods, 575-576

org.w3c.dom.Node object, fields, 574-575
out-of-line links (extended XLinks), 494
 linkbases, 497-498
 linksets, 497
output control, 509
output document type (XSLT), 327-328

P

<p> element, 398-399
packaging data, 13
page masters, 345-347
page sequences
 creating, 348-349
 flow, creating, 349-350
 <fo:root> element, 345
pages (Web), HTML sample, 10
<par> element, 480
<paragraph> element, syntax, 272
parameter entity references, 55-56
parameter DTD entities, 222
 % (percent sign), 161
 ; (semicolon), 161
 external, creating, 172-174
 internal, creating, 171-172
parameterized DTDs, 175
parent axis, 310
parent elements, 84
parsed character data, 56, 116
parsed entities, well-formed, 80
parsers, SAX, 612, 618
parsing
 documents, 70-73, 570
 entities, 73
<path> element, 467, 469
paths (SVG)
 creating, 467-468
 text, creating, 469-470
pattern schema facet, 209-210
#PCDATA (parsed character data), DTDs, 117
PDF documents
 creating with XSL-FO, 339-340
 viewing, 340
percent sign (%), 23, 161
Perl, regular expressions, 209
players, SOJA (SMIL Output in Java Applets), 482
plus sign (+), 17, 119-124
point() function, 503
points (XPointer schemes), 501-502
<polygon> element, 463
polygons (SVG), creating, 463-464
<polyline> element, 462
polylines (SVG), creating, 462-463
position property, 275
position() function, 313
positioning
 relative, 275-278
 styles, CSS, 274-277
preceding axis, 310
preceding-sibling axis, 310
predicates, 311
 boolean expressions, 312
 node sets, 313-315
 numbers, 315-316
 result tree fragments, 312
 strings, 316
presentations, separating data from (XForms), 510-512
private external DTDS, creating, 128-130
processing instructions, 18
 attributes, 65-70
 handling, 306, 580
 SAX, 615
 writing, 62
processing-instruction() node test, 311

processors

- CDATA sections, 70-73
- instructions, writing, 62
- Saxon XSLT, downloading, 286

productions, well-formed documents, 79**programming**

- handling, 776
 - documents, 766-767, 774-775
 - elements, creating, 771-773
 - schemas, 768-774
- objects, DOM, 535
- Visual Studio .NET, 766

Project menu commands

- Add Existing Item, 181
- Add Item, 766
- Set as StartUp Project, 792

prologs

- well-formed documents, 79
- writing, 58-59

properties

- background-attachment, 265, 271
- background-color, 265
- background-image, 266, 271
- background-positions, 271
- background-repeat, 266, 271
- border-bottom-width, 268
- border-color, 268
- border-left-width, 268
- border-right-width, 268
- border-style, 268
- border-top-width, 268
- border-width, 268
- bottom, 275
- color, 266
- controls, 787
- CSS, 253
- DataGrid class, 738-739
- DataSet class, 736
- display, 264
- DOMDocument object, 536-538

- <fo:block> element, 351-352
- <fo:external-graphic> element, 358-359
- <fo:inline> element, 354-355
- <fo:page-number> element, 361-362
- <fo:page-sequence> element, 348
- font-family, 264
- font-size, 264
- font-style, 264
- font-weight, 264
- left, 275
- line-height, 264, 269
- list-item, 278
- list-style-image, 278
- list-style-type, 278
- margin-bottom, 269
- margin-left, 270
- margin-right, 270
- margin-top, 270
- OleDbDataAdapter object, 735
- page masters, 345-347
- position, 275
- record sets, DSOs (data source objects), 701-702
- right, 275
- style, 265
- table, 279
 - table-caption, 279
 - table-cell, 279
 - table-column, 279
 - table-column-group, 279
 - table-footer-group, 279
 - table-header-group, 279
 - table-row, 279
 - table-row-group, 279
- text-align, 265, 270
- text-decoration, 265
- text-indent, 265, 270
- text-transform, 265
- top, 275
- ValidateOnParse, 560
- vertical-align, 265, 270

- XmlDataDocument object, 745
- XMLDOMAttribute object, 541-542
- XMLDOMElement object, 540
- XMLDOMNode, 539
- XMLDOMText object, 542-543
- XSL-FO, 343-344
- property elements, RDF (Resource Description Framework), 672**
- property/value pairs, rule specifications, 253**
- protocols, .NET (Microsoft, ADO.NET), 33.**
 - See also* SOAP (Simple Object Access Protocol)
- public external DTDS, creating, 130-131**
- PUBLIC keyword, 130, 165**

Q-R

- qualifying local element declarations, 228-232**
- Query Builder, 733**
- query languages. *See* XQuery**
- question mark (?), 119-124**
- quotation marks (""), 89**
- range control, 509**
- ranges (XPath), XPointer schemes, 501-504**
- <rdf:Description> element, 672**
- RDF (Resource Description Framework), 645, 668**
 - abbreviated syntax, 679-680
 - browser support, 669-670
 - descriptions
 - elements, 672
 - nesting, 676-678
 - Dublin Core language, 670
 - attributes, 674
 - elements, 673-674
 - resource types, 674-675
 - property elements, 672

- resources, 669
 - attributes, 677
 - multiple, 675-677
- root elements, 671
- statements, 670-671
- XML, storing data, 678-679
- RDF viewer, 670**
- Read XML Data button, 731**
- reading**
 - documents with Java, 568-572
 - attributes, 577-578
 - CDATA sections, 580
 - child nodes, 579
 - childLoop method, 573-576
 - document nodes, 576-577
 - elements, 577, 580-584
 - processing instructions, 580
 - text nodes, 579
 - documents with JavaScript, 544-546
 - attributes, 551-553
 - data islands, 547-549
 - finding by name, 549-551
 - syntax (Visual Basic .NET), 780-784
- record sets, DSOs (data source objects), 696**
 - methods, 702-703
 - properties, 701-702
- records, DSOs (data source objects), 696, 699-701**
- <rect> element, 456**
- rectangles (SVG), creating, 456**
- recursion, methods, 554**
- redefine schema element, 194**
- references, DTD entities**
 - general, 161-168
 - nesting, 165
- regions, page masters, 346-347**
- regular expressions, 209**
- rel attribute, 423-424**
- relationships, elements, 84**
- relative location paths, XPath, 309**

relative positioning, 275-278

remoting, 37

#REQUIRED default values, DTD attributes, 147-148

Reset buttons (XForms), 516-517

<reset> element, 516

Resource Description Framework. *See* RDF

resource types, Dublin Core language, 674-675

resources

attributes, RDF, 677

CSS, 252

multiple, RDF, 675-677

RDF (Resource Description Framework), 669

schemas, 180

SOAP (Simple Object Access Protocol), 646

tutorials, 39

Usenet newsgroups, 39

W3C Web site, 37-38

XHTML (Extensible Hypertext Markup Language), 30

restriction schema element, 194

result tree fragments (XPath), 312

Rich Text Format (RTF), files, 11-12

right property, 275

root elements, 16, 64

adding (well-formed documents), 80-81

RDF (Resource Description Framework), 671

well-formed documents, 79, 85-87

root nodes

matching, 301

trees, 293

round() function, 315

rows, tables (XHTML), 437-438

RTF (Rich Text Format), files, 11-12

rules, selectors, 252

S

saving documents, 44

SAX (Simple API for XML), 607

callback methods, 613

childLoop method, 611

DefaultHandler object, 611-614

documents

elements, finding by name, 623-628

navigating, 633-637

starting, 614

elements, 615-619

errors/warnings, handling, 619-623

overview, 608-610

parsers, 612, 618

processing instructions, 615

text, handling, 618

SAX/Java

browsers, creating, 628-633

XML, writing, 637-641

Saxon XSLT processor, downloading, 286

SAXParser class, methods, 612-613

SAXParserFactory object, 611-612

Scalable Vector Graphics. *See* SVG

schema element, 194

schemas, 25

all groups, declaring, 222

annotations, 192, 233-234

anonymous type definitions, 215-217

attributes

creating, 202-203

declaring, 198-200

grouping, 221-222

choices, creating, 214

complex types, 217-218, 770-771

creating, 189-191, 767

data types, specifying, 208

declarations, 192

documents, connecting to (Visual Basic .NET), 773-774

- elements, 193-194
 - declaring, 195, 198-200
 - default values, 201
 - global declarations, 223
 - grouping, 219-220
 - local declarations, 223
 - number specification, 200-201
 - type, 195-200
- empty elements, declaring, 217-218
- mixed-content elements, 218-219
- namespaces, 192, 222
 - local element declarations, 224-232
 - URIs, 94
- resources, 180
- simple types, creating (Visual Basic .NET), 768-770
- tools
 - HiT Software, 181
 - Microsoft Visual Studio .NET, 181-183
 - validators, 184-189
 - xmlArchitect, 181
 - XMLspy, 181
 - XRay, 181
- types
 - simple ordered, restricting, 212-213
 - simple, restricting, 208-213
- URIs, specifying, 186
- validators, 49
- schemas facets**
 - simple ordered schema types, 212-213
 - simple schema types, 208-213
- schemes**
 - element schemes (XPath, 499-500)
 - namespace schemes (XPath), 500
 - XPointer schemes, 500
 - points, 502-503
 - ranges, 503-504
 - XPath functions, 501
 - XPath location sets, 501
 - XPath locations, 501
 - XPath node tests, 501
 - XPath points, 501
 - XPath ranges, 501
- Scholarly Technology Group's XML validator, 112-113**
- SDK (Software Development Kit), 650**
- searches, XML data, DSOs/JavaScript, 716-719**
- secret control, 509**
- <select> element, 512**
- <select1> element, 512**
- <selectboolean> element, 515**
- select attribute, XPath, 309**
 - abbreviations, 317-321
 - axes, 310-311
 - default rules, 317-321
 - node tests, 311
 - predicates, 311-316
 - tools, 321
- select Booleans (XForms), creating, 515-516**
- select controls, 509, 512-513**
- selector schema element, 194**
- selectors**
 - rules, 252
 - style sheets (CSS)
 - creating, 256-257
 - grouping, 256
 - ID values, 261-262
 - style classes, 257-260
- self axis, 310**
- semicolon (;), 161, 263**
- sensitive characters, style sheets, 434**
- <seq> element, 480**
- sequence schema element, 194**
- sequences**
 - content models, 116, 119-122
 - multimedia, creating, 480
- server-side XSLT, 288, 290**
- servers**
 - SOAP, 650-662
 - Tomcat, 657-658, 665-666

servlets, SOAP (Java example), 656

Set as StartUp Project command (Project menu), 792

SGML (Standard Generalized Markup Language), 13

sibling elements, 83

simple anonymous types, schemas, 216

Simple API for XML. *See* SAX

Simple Object Access Protocol. *See* SOAP
simple ordered types (schemas), restricting, 12-213

simple schema element type, 195-197

simple types (schemas)

 creating, 768-770

 restricting, 208-213

simple XLinks

 Amaya Web browser, 488-489

 attributes, 489-490

 emulating HTML hyperlinks, 488-489

 implementing, 486-487

 Internet Explorer, 489

 xlink:actuate attribute, 493

 xlink:arcrole attribute, 493

 xlink:label attribute, 493

 xlink:ref attribute, 491

 xlink:role attribute, 493

 xlink:show attribute, 492

 xlink:title attribute, 493

 xlink:type attribute, 491

simpleContent schema element, 194

simpleType schema element, 194

sites. *See* Web sites

SMIL (Synchronized Multimedia Integration Language), 29-30

 documents, creating, 479-482

 multimedia sequences, creating, 480

SMIL Output in Java Applets (SOJA) Player, 482

SOAP (Simple Object Access Protocol), 36-37, 645

 attachments, adding, 659

 attributes, 649-650

 body, 647

 clients, creating, 653-655

 elements, 647-649

 envelopes, 647

 headers, 647

 Java example, 657

 clients, 662-668

 servers, creating, 658-662

 servlets, 656

 .NET example, 650

 servers/clients, 655-656

 SOAP, 651-655

 resources, 646

 servers, 650-653

 syntax, 646-647

SoapFormatter class, 651

SoapFormatter object, 653

software

 AchieveForms, 507

 Chiba, 507

 FormFaces, 507

 FormsPlayer, 506

 HiT Software, 181

 jXForms, 507

 LiquidOffice, 507

 Mosquito XForms, 506

 NMatrix, 507

 Novell XForms, 507

 TrustForm System, 507

 X-Smiles, 506

 Xero, 507

 XForms, 506-507

 XML Forms Package, 507

 XMLForm, 507

 XServerForms, 507

Software Development Kit (SDK), 650

SOJA (SMIL Output in Java Applets) Player, 482

** element, 405-406**

spanning columns, headers, 440

standalone attribute, 60, 80, 328, 387

Standard Generalized Markup Language (SGML), 13

starts-with(string1, string2) function, 316

statements, RDF (Resource Description Framework), 670-671

Store XML Data button, 731

storing

databases as XML, 731-743

XML data, RDF, 678-679

strict XHTML (Extensible Hypertext Markup Language), 31

Strict XHTML 1.0 DTD, 384

string matching, ranges (XPath schemes), 503

string-length(string1), 316

strings (XPath), 316

style attribute, 255, 395, 434-435

style classes, creating, 257-260

<style> element, 432-435

style properties, 265

style rule specification, 263

alignment styles, 269-270

background styles, 265-268

block-level elements, 264

border styles, 268-269

color styles, 265-268

curly braces {}, 252

image styles, 270-274

list styles, 278

margin styles, 269-270

positioning styles, 274-277

property/value pairs, 253

table styles, 279-281

text styles, 264-265

style rule specification, semicolon (;), 263

style rules, 252

style sheet selectors (CSS)

creating, 256-257

grouping, 256

ID values, 261-262

style classes, 257-260

style sheets, 252

CSS (Cascading Style Sheets), 18

data, displaying, 18

embedded, creating, 432-435

external, syntax, 425

files, viewing, 18-19

sensitive characters, 434

XSL (Extensible Stylesheet Language), 18

data extraction, 298

multiple matches, 300

syntax, 296

XSLT, 296-295, 786-787

styles

alignment, CSS, 269-270

background, CSS, 265-268

border, CSS, 268-269

color, CSS, 265-268

CSS, SVG documents, 457-459

font, 265

images, CSS, 270-274

inline

creating, 255, 434

CSS, 262-263

lists, CSS, 278

margin, CSS, 269-270

positioning, CSS, 274-277

tables, CSS, 279-281

text, CSS, 264-265

Submit buttons (XForms), 516-517

submit control, 509

<submit> element, 516

subsequences

content models, creating, 120

syntax (sample document), 120-122

subsets, external (DTDs), 171

substring(string1, offset, length), 316

substring-after(string1, string2), 316

substring-before(string1, string2), 316

sum() function, 315

SVG (Scalable Vector Graphics), 35, 450, 453

animations, creating, 472-474

attributes, 459

circles, creating, 460

colors, hexadecimal numbers, 454

documents, creating, 454-459

DOM (Document Object Model), 451

DTDs, 455

elements, 452

ellipses, creating, 461

embedding in HTML, 478

gradients, creating, 466-467

graphic objects, 451

groups, creating, 471-472

hyperlinks, creating, 474-475

JavaScript, creating, 476-478

lines, creating, 462

MIME type, 455

paths, 467-470

polygons, creating, 463-464

polylines, creating, 462-463

predefined colors, 453-454

rectangles, creating, 456

text, creating, 464-465

transformations, creating, 471-472

SVG Viewer, 456-457

.svg file extension, 455

symbols

content model sequences, 119-122

DTDs, 118, 124

Synchronized Multimedia Integration Language. See SMIL

syntax

abbreviated, RDF, 679-680

animations, creating, 473

<!ATTLIST> element, 142

border styles, 269

browsers, creating, 589, 593-595

circles, creating, 460

class-specific selectors, 259-260

CML (Chemical Markup Language), 29

color styles, 266

context code, XQuery, 753-757

CSS

documents, connecting, 254

inline styles, 262

sample document, 253

style sheet selectors, grouping, 257

data islands, data binding HTML elements to XML data, 704-705

databases, ASP (Active Server Pages), 729-730

declared attributes, DTDs (sample document), 143-144

documents

DTDs, adding, 561

editing, 601-604

qualified attribute, 232

qualified elements/attributes, 230

qualified locals, 229

sample (ch08_01.xml), 250-251

sample (ch10_01.xml), 334

unqualified locals, 227-228

verifying, 186-188

viewing via style sheets, 19

DSO (data source object) applet, 710-716

DTDs (Document Type Definition), 25

choices (sample document), 122-123

entities, declaring, 163-164

namespaces, 134-135

sample document, 110-111

ellipses, creating, 461

empty elements, content models (DTDs), 126-127

external DTDs, 132

external general entity references (DTDs), 166-167

- external parameter DTD entities, 173-174
- external style sheets, 425
- font styles, 265
- foreground images, 273-276
- gradients, creating, 467
- hierarchical XML data, 720-721, 724-725
- HTML documents
 - data holding, 694-695
 - document validation, 185-188
 - sample, 382
 - sample Web page, 10
- HTML DSO document, 700-701
- HTML+TIME, 32
- hyperlinks, creating, 475
- ID-based selectors, 261
- image styles, 271
- internal parameter DTD entities, 171-172
- internal/external DTDs (sample document), 132
- Java
 - document elements, finding by name, 585-588
 - documents, navigating, 596-597
 - files, data extraction, 22-23
 - parsing documents, 581-583
 - programs, 292
 - SOAP, 660-665
- Java/SAX
 - browsers, creating, 631-633
 - documents, navigating, 637-641
- JavaScript
 - creating, 477
 - data islands, 548-549
 - documents, attributes, 552-553
 - documents, extracting data, 558-560
 - documents, finding elements by name, 550
 - documents, validating with DTDs, 562-563
 - files, data extraction, 21
 - reading documents, 546-547
- JSP, server-side XSLT, 289
- lines, creating, 462
- local namespaces, 97-98
- margin styles, 270
- MathML (Mathematical Markup Language)
 - file, 28
- mixed content models (sample document), 124-125
- multiple declared attributes, DTDs (sample documents), 144-145
- <name> element, 299
- namespaces, 96
- nesting errors, 24-25
- <paragraph> element, 272
- paths, creating, 468
- polygons, creating, 464
- polylines, creating, 463
- predefined general entity references, DTDs (sample document), 161
- private external DTDs (sample document), 129
- public external DTDs (sample document), 131
- RDF, 675-678
- rectangles, creating, 456
- relative positioning, 276-278
- RTF files, example, 11-12
- SAX
 - browsers, creating, 635-637
 - document elements, finding by name, 625-628
 - documents, parsing, 620-623
 - sample, 610
- schemas
 - creating (sample document), 189-190
 - example, 187
 - qualified attribute, 231
 - unqualified locals, 226-227
 - validating (sample document), 190-191

SMIL (Synchronized Multimedia Integration Language), 29, 481
SOAP (Simple Object Access Protocol), 646-647, 650-654
<style> element, 435
style classes, 257-258
subsequences (sample document), 120-122
SVG (Scalable Vector Graphics), 35, 458, 478
table styles, 280
text, creating, 465
textpaths, creating, 470
transformations, creating, 471
Visual Studio .NET
 reading, 780-784
 writing, 777-780
well-formed documents, creating, 82-83
XForms, 505-506
XHTML (Extensible Hypertext Markup Language) file, 31
 <!-- -> element, 415
 <a> element, 421
 element, 409
 <center> element, 401-402
 <div> element, 403-404
 extending, 444-446
 element, 413-414
 <frame> element, 430-431
 heading elements, 407-408
 element, 427
 <link> element, 424
 paragraphs/line breaks, 399
 sample, 383
 element, 405-406
 <style> element, 433-434
 style sheets, 396
 <td> element, 442-443
XHTML and MathML, 100-101
XML data
 displaying in HTML tables, 707-708
 document (data binding), 703-704
 searches, DSOs/JavaScript, 718-719

XML sample file, 14
XQuery document, 759-760
XSL style sheets
 abbreviations, 318-319
 data extraction, 298
 multiple matches, 300, 303-308
 position() function, 313-314
 sample, 296
 <xsl:choose> element, 325-326
 <xsl:copy> element, 322
 <xsl:if> element, 323-324
XSL-FO
 creating with XSLT, 335-339
 <fo:inline> element, 355-357
 <fo:page-number> element, 360
 lists, creating, 363-364
 tables, creating, 366-368
XSLT documents, images, 358
XSLT style sheet, 786-787
SYSTEM keyword, 128, 165

T

<table> element, 435-437, 706
table formatting (XSL-FO), 365-368
table property, 279
table styles, CSS, 279-281
table-caption property, 279
table-cell property, 279
table-column property, 279
table-column-group property, 279
table-footer-group property, 279
table-header-group property, 279
table-row property, 279
table-row-group property, 279
tables
 data (XHTML), 440-443
 formatting (XHTML), 435-437
 headers (XHTML), 438-440

- HTML, 706-709, 714-716
- rows (XHTML), 437-438
- tabular data control (TDC), 694**
- tags. *See also* namespaces**
 - closing, elements, 15
 - reading documents, 580-584
 - SAX, 619
 - HTML, 10-12
 - naming, 63
 - opening (elements), 15
- targetNamespace attribute, 223**
- <td> element, 440-443**
- TDC (tabular data control), 694**
- template files, XQuery, 757-758**
- templates**
 - master, creating, 345
 - XSLT, 294
 - node matching, 301-308
 - <xsl:apply-template> element, 295-298
 - <xsl:for-each> element, 298-300
 - <xsl:value-of> element, 298-300
- text**
 - non-markup, DTDs, 116
 - SAX, handling, 618
 - SVG, creating, 464-465
 - XHTML, 397
 -
 element, 400
 - <center> element, 400-402
 - <div> element, 402-405
 - <h1> to <h6> elements, 406-408
 - <p> element, 398-399
 - element, 405-406
- text attribute, 395**
- text boxes, multiline, 743**
- text content handling, element content models, DTDs, 116-118**
- text data, mixed content models (DTDs), 124**
- <text> element, 464**
- text editors**
 - Adobe FrameMaker, 44
 - Visual Studio XML designer, 45-46
 - XML Notepad, 44-46
 - XML Pro, 44
 - XML Spy, 45
 - XML Writer, 44-45
 - XMLmind, 45
- <textpath> element, 469-470**
- text formatting (XHTML)**
 - <!-- -> element, 415
 - element, 408-410
 - element, 411-414
 - <i> element, 410-411
 - <u> element, 411
- text nodes, 579**
- text paths (SVG), creating, 469-470**
- text styles, CSS, 264-265**
- text() node test, 311**
- text-align property, 265, 270**
- text-decoration property, 265**
- text-indent property, 265, 270**
- text-transform property, 265**
- textarea control, 509**
- <th> element, 438-440**
- title attribute, 395**
- <title> element (document title), 393-394**
- titles, SVG documents, 455-456**
- Tomcat server, 655-658**
- tools**
 - schemas
 - HiT Software, 181
 - Microsoft Visual Studio .NET, 181-183
 - validators, 184-189
 - xmlArchitect, 181
 - XMLspy, 181
 - XRay, 181
 - XPath, 321
- Tools menu commands, Connect to Database, 732**

top property, 275

Topologi Schematron Validator, 184

totalDigits schema facet, 210

ToUpper method, 651

<tr> element, 437-438

transactional XHTML (Extensible Hypertext Markup Language), 30-31

transform attribute, 471

transforming documents, XSLT, 286

clients, 290-291

Java, 291-292

servers, 288-290

transformations (SVG), creating, 471-472

Transitional XHTML 1.0 DTD, 384

translate(string1, string2, string3), 316

trees, nodes, 293

<trigger> element, 514

trigger control, 509

TrustForm System, 507

tutorials, 39

type, schema elements

complex, 195, 198-200

simple, 195-197

type values, DTD attributes, 145-146

CDATA, 150-151

ENTITIES, 157-158

ENTITY, 156-157

enumerated, 151-152

ID, 154-155

IDREF, 155-156

NMTOKEN, 152-153

NMTOKENS, 153

NOTATION, 158-160

types

complex, schemas, 217-218

data (schemas), specifying, 208

simple (schemas), restricting, 208-213

simple ordered (schemas), restricting,
212-213

U

<u> element, 411

UCS (Universal Character System), 52

**UCS Transformation Format-8 (UTF-8), 14,
52**

UCS Transformation Format-16 (UTF-16), 53

underline element, text (XHTML), 411

Unicode, 52-54

**Uniform Resource Identifiers (URIs), name-
spaces, 93-97**

union schema element, 194

unique schema element, 194

Universal Character System (UCS), 52

unparsed entities, 74

**unparsed external DTD entities, declaring,
168**

upload control, 509

upper method, 651-654

URIs (Uniform Resource Identifiers)

external general entity references (DTDs),
165

namespaces, defining, 93-97

private external DTDs, 130

schemas, 186

XBase, 504

URLs, private external DTDs, 130. *See also*

URIs

use attribute, 202-203

Usenet newsgroups, 39

**UTF-8 (UCS Transformation Format-8), 14,
52**

UTF-16 (UCS Transformation Format-16), 53

V

**valid documents. *See also* DTDs (document
type definitions); schemas**

valid files, 24-26

validateOnParse property, 560

validating

- data, 775
- documents with DTDs (JavaScript), 560-564
- SVG documents, 455
- XHTML documents, 390-391

validators, 48-49, 112

- CSS, 253
- schemas, 184-189

value attribute, 203

values. See default values

variables, XQuery, 754

Vector Markup Language (VML), 450-451

version attribute, 15, 59, 84, 328

vertical-align property, 265, 270

viewers, RDF (Resource Description

Framework), 670

viewing

- files, 17-20
- PDF documents, 340
- SVG documents, SVG Viewer, 456-457

Visual Basic .NET applications, databases, 731-732

Visual Studio XML designer, 45-46

Visual Studio .NET programming

- controls, displaying formatted XML, 784-789
- data, handling, 776
- documents, 766-767, 774-775
- elements, creating, 771-773
- schemas
 - complex types, creating, 770-771
 - connecting to documents, 773-774
 - simple types, creating, 768-770
- syntax, 777-784
- Web services, 789-797

vlink attribute, 395

VML (Vector Markup Language), 450-451

W

W3C (World Wide Web Consortium), 13, 749-750

W3C DOM. See DOM (Document Object Model)

W3C Web site, 37-38

warning method, 619

warnings, SAX, handling, 619-623

Web applications, creating, 787

Web browsers

- Amaya, 27, 488-489
- creating, 589-595, 628-633
- DTDs, 113
- files, viewing, 17-20
- Internet Explorer, 47, 67, 72, 489
- Jumbo, 28, 48
- Netscape Navigator, 47
- support, RDF (Resource Description Framework), 669-670

Web pages, HTML sample, 10

Web services (Visual Studio .NET)

- calling, 792-797
- creating, 789-792

Web sites

- AchieveForms, 507
- Adobe, 340
- Chiba, 507
- FormFaces, 507
- FormsPlayer, 506
- HiT Software, 181
- jXForms, 507
- LiquidOffice, 507
- Microsoft, 181, 184
- Mosquito XForms, 506
- NMatrix, 507
- Novell XForms, 507
- TrustForm System, 507
- W3C, 37-38

- Xero, 507
- XML Forms Package, 507
- xmlArchitect, 181
- XLinks
 - attributes, 489-493
 - arcs, 495-497
 - emulating HTML hyperlinks, 488-489
 - extended XLinks, 494
 - implementing, 486-487
 - linkbases, 497-498
- XMLForm, 507
- XMLspy, 181
- XRay, 181
- XServerForms, 507
- X-Smiles, 506
- well-formed documents, 24-25**
 - creating, 80-84
 - defined, 78
 - productions, document, 79
 - prologs, 79
 - root elements, 79-81
 - standalone attribute, adding, 80
 - well-formed parsed entities, 80
 - well-formedness constraints, 80
 - attributes, 88-89
 - declarations, 84
 - element structure, 85-86
 - entity references, 89-92
 - legal character references, 85
 - nesting elements, 87-88
 - root elements, 85-87
- well-formed parsed entities, 80**
- well-formedness constraints, 80**
 - attributes, 88-89
 - declarations, 84
 - element structure, 85-86
 - entity references, 89-92
 - legal character references, 85
 - nesting elements, 87-88
 - root elements, 85-87
- While loop, 781**
- whitespace**
 - DTD attributes, 153
 - Java, 596
 - SAX parsers, 618
 - XML documents, 57
- white Space schema facet, 210**
- width attribute, 456**
- wildcard character**
 - *, 302
 - @*, 305
- Windows Application icon, 792**
- wizards, Data Adapter Configuration Wizard, 732, 744, 791**
- WordPad, 44**
- working drafts (W3C file), 13**
- World Wide Web Consortium (W3C), 13**
- WriteStartDocument method, 778**
- WriteStartElement method, 778**
- writing**
 - attributes, 65-68
 - comments, 60-61
 - declarations, 59-60
 - documents, 51
 - attributes, 65-70
 - CDATA sections, 70-73
 - character encodings, 52-54
 - comments, 60-61
 - declarations, 59-60
 - elements, naming tags, 63
 - empty elements, 64
 - entities, 55-56, 73-74
 - line endings, 58
 - markups, 55-56
 - processing instructions, 62
 - prologs, 58-59
 - root elements, 64
 - whitespace, 57
 - processing instructions, 62
 - prologs, 58-59

syntax (Visual Basic .NET), 777-780
 XForms, 509-510
 XML, 597-604, 637-641

X-Z

X-Smiles, 506

XBase, 504

Xerces, 184

Xero, 507

XForms, 504-505

buttons, 513-517
 controls, 508-509, 512-513
 declarations, 509
 presentations, separating data from, 510-512
 select Booleans, creating, 515-516
 software, 506-507
 writing, 509-510

XHTML (Extensible Hypertext Markup Language), 30-31, 381-384

<a> element, 419-422
 <body> element (document body), 394-397
 documents
 <!DOCTYPE> element, 387-389
 columns, 428-429
 declarations, 387
 requirements, 386
 standalone attributes, 387
 validating, 390-391
 writing, 386-390
 extending, 443-446
 <frame> element, 429-432
 <frameset> element, 427-429
 <head> element (document head), 392-393
 headers, columns, spanning, 440
 <html> element (document element), 391-392
 element, 425-427

inline styles, creating, 434
 <link> element, 422-425
 mixed-content models, 397
 <style> element, 432-435
 <table> element, 435-437
 <td> element, 440-443
 text, 397

 element, 400
 <center> element, 400-402
 <div> element, 402-405
 <h1> to <h6> elements, 406-408
 <p> element, 398-399
 element, 405-406
 text formatting
 <!--> element, 415
 element, 408-410
 element, 411-414
 <i> element, 410-411
 <u> element, 411
 <th> element, 438-440
 <title> element (document title), 393-394
 <tr> element, 437-438

XHTML 1.0, 384-385, 388

XHTML 1.1, 385, 388

XHTML 2.0, 385-386, 389

XHTML Basic, 386, 389

XHTML validator, 390

XHTML/MathML document, 100-101

XLinks

Amaya Web browser, 488-489
 attributes, 489-490
 xlink:actuate attribute, 493
 xlink:arcrole attribute, 493
 xlink:label attribute, 493
 xlink:ref attribute, 491
 xlink:role attribute, 493
 xlink:show attribute, 492
 xlink:title attribute, 493
 xlink:type attribute, 491

- extended XLinks
 - arcs, 495-497
 - inline links, 494
 - linkbases, 497-498
 - out-of-line links, 494, 497-498
- HTML hyperlinks, emulating, 488-489
- implementing, 486-487
- Internet Explorer, 489
- xlink:actuate attribute, 493**
- xlink:arcrole attribute, 493**
- xlink:label attribute, 493**
- xlink:ref attribute, 491**
- xlink:role attribute, 493**
- xlink:show attribute, 492**
- xlink:title attribute, 493**
- xlink:type attribute, 491**
- XML**
 - applications, 26-27
 - declarations, 14-15
 - processors, 24
 - schemas, 25
- <?xml-stylesheet?> processing instruction, 254-256**
- <XML> element, 548**
- XML Notepad, 44-46**
- XML Pro, 44**
- XML Spy, 45**
- XML Writer, 44-45**
- xml:lang attribute, 69-70**
- xml:space attribute, 57**
- XMLmind, 45**
- XML Forms Package, 507**
- XML Path Language (XPath), 293**
- XML Schema Quality Checker, 184**
- xmlArchitect Web sites, 181**
- XmlDataDocument object**
 - methods, 746-747
 - properties, 745
- XMLDOMAttribute object, 541-542**
- XMLDOMElement object, 540-541**
- XMLDOMNode object, 539-540**
- XMLDOMText object, 542-544**
- XMLForm, 507**
- xmlns attribute, 98**
- xmlns:prefix attribute, 93, 97**
- XMLspy Web site, 181**
- XmlTextReader object, 780**
- XmlTextWriter class, 777**
- xml:lang attribute, 395**
- XPath (XML Path Language), 293**
 - abbreviations, 317-321
 - axes, 310-311
 - databases, 743-749
 - default rules, 317-321
 - expressions, 308-309
 - location steps, 309
 - node tests, 311
 - predicates, 311-316
 - tools, 321
 - XPointer schemes, 500-504
- XPath Visualiser, 321**
- XPointers**
 - barenames, 499
 - element schemes, 499-500
 - framework specifications, 499
 - namespace schemes, 500
 - software support, 498
 - XPointer schemes, 500
 - points, 502-503
 - ranges, 503-504
 - XPath functions, 501
 - XPath location sets, 501
 - XPath locations, 501
 - XPath node tests, 501
 - XPath points, 501
 - XPath ranges, 501
- XQuery**
 - context code, 753-757
 - databases
 - implementations, 750-752
 - Lucent Galax XQuery processor, 752-761

- online working drafts, 750
- W3C, 749-750
- functions, 753-754, 758-759
- template files, creating, 757-758
- variables, 754
- XRay Web site, 181**
- <xsd:annotation> element, 233**
- <xsd:appInfo> element, 233-234**
- <xsd:documentation> element, 233-234**
- <xsd:element>, 195**
- XSD Schema Validator, 184**
- XServerForms, 507**
- XSL (Extensible StyleSheet Language), 18**
 - data extraction, 298
 - multiple matches, 300
 - <xsl:apply-templates> element, 295-298**
 - <xsl:choose> element, 324-328**
 - <xsl:copy> element, 321-322**
 - <xsl:for-each> element, 298-300**
- XSL Formatting Objects. *See* XSL-FO**
- <xsl:if> element, 323-324**
- <xsl:output> element, 327**
- XSL style sheet, syntax, 296**
- XSL-FO (XSL Formatting Objects), 331, 334**
 - documents, creating, 335-339
 - <fo:block> element, 333**
 - inline formatting, 353
 - <fo:external-graphic> element, 357-359**
 - <fo:inline> element, 354-357**
 - <fo:page-number> element, 360-362**
 - list formatting, 362-364
 - namespaces, 332
 - objects, 341-343
 - <fo:block> element, 350-353**
 - <fo:flow> element, 349-350**
 - <fo:layout-master-set> element, 345**
 - <fo:page-sequence > element, 348-349**
 - <fo:region-after> element, 347-348**
 - <fo:region-body> element, 347-348**
 - <fo:root> element, 344-345**
 - <fo:simple-page-master> element, 345-347**
 - <fo:static-content> element, 349**
 - PDF documents, creating, 339-340
 - properties, 343-344
 - table formatting, 365-368
- XSLT (Extensible Stylesheet Language Transformation), 285-286**
 - documents, transforming, 286-292
 - matching, XPath expressions, 308-309
 - nodes, copying, 321
 - output document type, 327-328
 - select attribute, XPath, 309
 - abbreviations, 317-321
 - axes, 310-311
 - default rules, 317-321
 - node tests, 311
 - predicates, 311-316
 - tools, 321
 - style sheets
 - writing, 293-295
 - XSL-FO documents, creating, 337-339
 - templates, 294
 - node matching, 301-308
 - <xsl:apply-templates> element, 295-298**
 - <xsl:for-each> element, 298-300**
 - <xsl:value-of> element, 298-300**
 - <xsl:choose> element, 324-326**
 - <xsl:copy> element, 321-322**
 - XSL-FO documents, creating, 335-336
 - <xsl:if> element, 323-324**
- XSLT style sheets, 786-787**
- <xsl:value-of> element, 298-300**
- XSV, 184**