

# **Definitive XML Schema**

**Second Edition**



## The Charles F. Goldfarb Definitive XML Series

### **Priscilla Walmsley**

- Definitive XML Schema Second Edition

### **Charles F. Goldfarb and Paul Prescod**

- Charles F. Goldfarb's XML Handbook™  
Fifth Edition

### **Rick Jelliffe**

- The XML and SGML Cookbook:  
Recipes for Structured Information

### **Charles F. Goldfarb, Steve Pepper, and Chet Ensign**

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

### **G. Ken Holman**

- Definitive XSL-FO
- Definitive XSLT and XPath

### **Bob DuCharme**

- XML: The Annotated Specification
- SGML CD

### **Truly Donovan**

- Industrial-Strength SGML:  
An Introduction to Enterprise Publishing

### **Lars Marius Garshol**

- Definitive XML Application Development

### **JP Morgenthal with Bill la Forge**

- Enterprise Application Integration with  
XML and Java

### **Michael Leventhal, David Lewis, and Matthew Fuchs**

- Designing XML Internet Applications

### **Adam Hocek and David Cuddihy**

- Definitive VoiceXML

### **Dmitry Kirsanov**

- XSLT 2.0 Web Development

### **Yuri Rubinsky and Murray Maloney**

- SGML on the Web:  
Small Steps Beyond HTML

### **David Megginson**

- Structuring XML Documents

### **Sean McGrath**

- XML Processing with Python
- XML by Example:  
Building E-commerce Applications
- ParseMe.1st:  
SGML for Software Developers

### **Chet Ensign**

- \$GML: The Billion Dollar Secret

### **Ron Turner, Tim Douglass, and Audrey Turner**

- ReadMe.1st:  
SGML for Writers and Editors

### **Charles F. Goldfarb and Priscilla Walmsley**

- XML in Office 2003:  
Information Sharing with Desktop XML

### **Michael Floyd**

- Building Web Sites with XML

### **Fredrick Thomas Martin**

- TOP SECRET Intranet:  
How U.S. Intelligence Built Intelink—The  
World's Largest, Most Secure Network

### **J. Craig Cleaveland**

- Program Generators with XML and Java

### **About the Series Author**

Charles F. Goldfarb is the father of XML technology. He invented SGML, the Standard Generalized Markup Language on which both XML and HTML are based. You can find him on the Web at: [www.xmlbooks.com](http://www.xmlbooks.com).

### **About the Series Logo**

The rebus is an ancient literary tradition, dating from 16th century Picardy, and is especially appropriate to a series involving fine distinctions between markup and text, metadata and data. The logo is a rebus incorporating the series name within a stylized XML comment declaration.

# **Definitive XML Schema**

**Second Edition**

■ Priscilla Walmsley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco  
New York • Toronto • Montreal • London • Munich • Paris • Madrid  
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Titles in this series are produced using XML, SGML, and/or XSL. XSL-FO documents are rendered into PDF by the XEP Rendering Engine from RenderX: [www.renderx.com](http://www.renderx.com).

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales  
(800) 382-3419  
[corpsales@pearsontechgroup.com](mailto:corpsales@pearsontechgroup.com)

For sales outside the United States, please contact:

International Sales  
[international@pearsoned.com](mailto:international@pearsoned.com)

Visit us on the Web: [informit.com](http://informit.com)

Library of Congress Cataloging-in-Publication Data is on file

Copyright © 2013 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-132-88672-7

ISBN-10: 0-132-88672-3

Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, MI.

First printing: September 2012

Editor-in-Chief: Mark L. Taub  
Managing Editor: Kristy Hart  
Book Packager: Alina Kirsanova  
Cover Designer: Alan Clements

*To Doug, my SH*

*This page intentionally left blank*

# Overview

<b>Chapter 1</b>	<b>Schemas: An introduction</b>	2
<b>Chapter 2</b>	<b>A quick tour of XML Schema</b>	16
<b>Chapter 3</b>	<b>Namespaces</b>	34
<b>Chapter 4</b>	<b>Schema composition</b>	56
<b>Chapter 5</b>	<b>Instances and schemas</b>	78
<b>Chapter 6</b>	<b>Element declarations</b>	88
<b>Chapter 7</b>	<b>Attribute declarations</b>	112
<b>Chapter 8</b>	<b>Simple types</b>	128
<b>Chapter 9</b>	<b>Regular expressions</b>	158
<b>Chapter 10</b>	<b>Union and list types</b>	180
<b>Chapter 11</b>	<b>Built-in simple types</b>	200
<b>Chapter 12</b>	<b>Complex types</b>	256
<b>Chapter 13</b>	<b>Deriving complex types</b>	300
<b>Chapter 14</b>	<b>Assertions</b>	350
<b>Chapter 15</b>	<b>Named groups</b>	384
<b>Chapter 16</b>	<b>Substitution groups</b>	406
<b>Chapter 17</b>	<b>Identity constraints</b>	422

<b>Chapter 18</b>	<b>Redefining and overriding schema components</b>	<b>446</b>
<b>Chapter 19</b>	<b>Topics for DTD users</b>	<b>472</b>
<b>Chapter 20</b>	<b>XML information modeling</b>	<b>500</b>
<b>Chapter 21</b>	<b>Schema design and documentation</b>	<b>538</b>
<b>Chapter 22</b>	<b>Extensibility and reuse</b>	<b>594</b>
<b>Chapter 23</b>	<b>Versioning</b>	<b>616</b>
<b>Appendix A</b>	<b>XSD keywords</b>	<b>648</b>
<b>Appendix B</b>	<b>Built-in simple types</b>	<b>690</b>

# Contents

<b>Foreword</b>	<b>xxxix</b>
<b>Acknowledgments</b>	<b>xxxiii</b>
<b>How to use this book</b>	<b>xxxv</b>
<b>Chapter 1 Schemas: An introduction</b>	<b>2</b>
<b>1.1   What is a schema?</b>	<b>3</b>
<b>1.2   The purpose of schemas</b>	<b>5</b>
1.2.1 <i>Data validation</i>	5
1.2.2 <i>A contract with trading partners</i>	5
1.2.3 <i>System documentation</i>	6
1.2.4 <i>Providing information to processors</i>	6
1.2.5 <i>Augmentation of data</i>	6
1.2.6 <i>Application information</i>	6
<b>1.3   Schema design</b>	<b>7</b>
1.3.1 <i>Accuracy and precision</i>	7
1.3.2 <i>Clarity</i>	8
1.3.3 <i>Broad applicability</i>	8

<b>1.4   Schema languages</b>	9
<b>1.4.1 Document Type Definition (DTD)</b>	9
<b>1.4.2 Schema requirements expand</b>	10
<b>1.4.3 W3C XML Schema</b>	11
<b>1.4.4 Other schema languages</b>	12
1.4.4.1 RELAX NG	12
1.4.4.2 Schematron	13
<b>Chapter 2 A quick tour of XML Schema</b>	<b>16</b>
<b>2.1   An example schema</b>	17
<b>2.2   The components of XML Schema</b>	18
2.2.1 <i>Declarations vs. definitions</i>	18
2.2.2 <i>Global vs. local components</i>	19
<b>2.3   Elements and attributes</b>	20
2.3.1 <i>The tag/type distinction</i>	20
<b>2.4   Types</b>	21
2.4.1 <i>Simple vs. complex types</i>	21
2.4.2 <i>Named vs. anonymous types</i>	22
2.4.3 <i>The type definition hierarchy</i>	22
<b>2.5   Simple types</b>	23
2.5.1 <i>Built-in simple types</i>	23
2.5.2 <i>Restricting simple types</i>	24
2.5.3 <i>List and union types</i>	24
<b>2.6   Complex types</b>	25
2.6.1 <i>Content types</i>	25
2.6.2 <i>Content models</i>	26
2.6.3 <i>Deriving complex types</i>	27
<b>2.7   Namespaces and XML Schema</b>	28

<b>2.8   Schema composition</b>	29
<b>2.9   Instances and schemas</b>	30
<b>2.10   Annotations</b>	31
<b>2.11   Advanced features</b>	32
2.11.1 <i>Named groups</i>	32
2.11.2 <i>Identity constraints</i>	32
2.11.3 <i>Substitution groups</i>	32
2.11.4 <i>Redefinition and overriding</i>	33
2.11.5 <i>Assertions</i>	33
<b>Chapter 3 Namespaces</b>	<b>34</b>
<b>3.1   Namespaces in XML</b>	35
3.1.1 <i>Namespace names</i>	36
3.1.2 <i>Namespace declarations and prefixes</i>	37
3.1.3 <i>Default namespace declarations</i>	39
3.1.4 <i>Name terminology</i>	40
3.1.5 <i>Scope of namespace declarations</i>	41
3.1.6 <i>Overriding namespace declarations</i>	42
3.1.7 <i>Undeclaring namespaces</i>	43
3.1.8 <i>Attributes and namespaces</i>	44
3.1.9 <i>A summary example</i>	46
<b>3.2   The relationship between namespaces and schemas</b>	48
<b>3.3   Using namespaces in schemas</b>	48
3.3.1 <i>Target namespaces</i>	48
3.3.2 <i>The XML Schema Namespace</i>	50
3.3.3 <i>The XML Schema Instance Namespace</i>	51
3.3.4 <i>The Version Control Namespace</i>	51

<b>3.3.5</b>	<i>Namespace declarations in schema documents</i>	52
3.3.5.1	Map a prefix to the XML Schema Namespace	52
3.3.5.2	Map a prefix to the target namespace	53
3.3.5.3	Map prefixes to all namespaces	54
<b>Chapter 4</b>	<b>Schema composition</b>	<b>56</b>
<b>4.1</b>	Modularizing schema documents	57
<b>4.2</b>	Defining schema documents	58
<b>4.3</b>	Combining multiple schema documents	61
<b>4.3.1</b>	<i>include</i>	62
4.3.1.1	The syntax of includes	63
4.3.1.2	Chameleon includes	65
<b>4.3.2</b>	<i>import</i>	66
4.3.2.1	The syntax of imports	67
4.3.2.2	Multiple levels of imports	70
4.3.2.3	Multiple imports of the same namespace	72
<b>4.4</b>	Schema assembly considerations	75
<b>4.4.1</b>	<i>Uniqueness of qualified names</i>	75
<b>4.4.2</b>	<i>Missing components</i>	76
<b>4.4.3</b>	<i>Schema document defaults</i>	77
<b>Chapter 5</b>	<b>Instances and schemas</b>	<b>78</b>
<b>5.1</b>	Using the instance attributes	79
<b>5.2</b>	Schema processing	81
<b>5.2.1</b>	<i>Validation</i>	81
<b>5.2.2</b>	<i>Augmenting the instance</i>	82
<b>5.3</b>	Relating instances to schemas	83
<b>5.3.1</b>	<i>Using hints in the instance</i>	84
5.3.1.1	The <code>xsi:schemaLocation</code> attribute	84
5.3.1.2	The <code>xsi:noNamespaceSchemaLocation</code> attribute	86
<b>5.4</b>	The root element	87

<b>Chapter 6 Element declarations</b>	<b>88</b>	
<b>6.1   Global and local element declarations</b>	89	
6.1.1 <i>Global element declarations</i>	89	■
6.1.2 <i>Local element declarations</i>	93	■
6.1.3 <i>Design hint: Should I use global or local element declarations?</i>	95	■
<b>6.2   Declaring the types of elements</b>	96	■
<b>6.3   Qualified vs. unqualified forms</b>	98	■
6.3.1 <i>Qualified local names</i>	98	
6.3.2 <i>Unqualified local names</i>	98	
6.3.3 <i>Using elementFormDefault</i>	99	
6.3.4 <i>Using form</i>	100	
6.3.5 <i>Default namespaces and unqualified names</i>	101	
<b>6.4   Default and fixed values</b>	101	
6.4.1 <i>Default values</i>	102	
6.4.2 <i>Fixed values</i>	103	
<b>6.5   Nils and nillability</b>	105	
6.5.1 <i>Using xsi:nil in an instance</i>	108	
6.5.2 <i>Making elements nillable</i>	109	
<b>Chapter 7 Attribute declarations</b>	<b>112</b>	
<b>7.1   Attributes vs. elements</b>	113	■
<b>7.2   Global and local attribute declarations</b>	115	■
7.2.1 <i>Global attribute declarations</i>	115	■
7.2.2 <i>Local attribute declarations</i>	117	■
7.2.3 <i>Design hint: Should I use global or local attribute declarations?</i>	119	■
<b>7.3   Declaring the types of attributes</b>	120	

<b>7.4</b>   Qualified vs. unqualified forms	122
<b>7.5</b>   Default and fixed values	123
<b>7.5.1</b> <i>Default values</i>	124
<b>7.5.2</b> <i>Fixed values</i>	125
<b>7.6</b>   Inherited attributes	126
<b>Chapter 8 Simple types</b>	<b>128</b>
<b>8.1</b>   Simple type varieties	129
<b>8.1.1</b> <i>Design hint: How much should I break down my data values?</i>	130
<b>8.2</b>   Simple type definitions	131
<b>8.2.1</b> <i>Named simple types</i>	131
<b>8.2.2</b> <i>Anonymous simple types</i>	132
<b>8.2.3</b> <i>Design hint: Should I use named or anonymous types?</i>	133
<b>8.3</b>   Simple type restrictions	135
<b>8.3.1</b> <i>Defining a restriction</i>	136
<b>8.3.2</b> <i>Overview of the facets</i>	137
<b>8.3.3</b> <i>Inheriting and restricting facets</i>	139
<b>8.3.4</b> <i>Fixed facets</i>	140
8.3.4.1 <i>Design hint: When should I fix a facet?</i>	141
<b>8.4</b>   Facets	142
<b>8.4.1</b> <i>Bounds facets</i>	142
<b>8.4.2</b> <i>Length facets</i>	143
8.4.2.1 <i>Design hint: What if I want to allow empty values?</i>	143
8.4.2.2 <i>Design hint: What if I want to restrict the length of an integer?</i>	144
<b>8.4.3</b> <i>totalDigits and fractionDigits</i>	145
<b>8.4.4</b> <i>Enumeration</i>	145
<b>8.4.5</b> <i>Pattern</i>	148
<b>8.4.6</b> <i>Assertion</i>	150

8.4.7	<i>Explicit Time Zone</i>	150
8.4.8	<i>Whitespace</i>	151
8.5	Preventing simple type derivation	152
8.6	Implementation-defined types and facets	154
8.6.1	<i>Implementation-defined types</i>	154
8.6.2	<i>Implementation-defined facets</i>	155
<b>Chapter 9 Regular expressions</b>		<b>158</b>
9.1	The structure of a regular expression	159
9.2	Atoms	161
9.2.1	<i>Normal characters</i>	162
9.2.2	<i>The wildcard escape character</i>	164
9.2.3	<i>Character class escapes</i>	164
9.2.3.1	Single-character escapes	165
9.2.3.2	Multicharacter escapes	166
9.2.3.3	Category escapes	167
9.2.3.4	Block escapes	170
9.2.4	<i>Character class expressions</i>	171
9.2.4.1	Listing individual characters	171
9.2.4.2	Specifying a range	172
9.2.4.3	Combining individual characters and ranges	173
9.2.4.4	Negating a character class expression	173
9.2.4.5	Subtracting from a character class expression	174
9.2.4.6	Escaping rules for character class expressions	175
9.2.5	<i>Parenthesized regular expressions</i>	175
9.3	Quantifiers	176
9.4	Branches	177
<b>Chapter 10 Union and list types</b>		<b>180</b>
10.1	Varieties and derivation types	181
10.2	Union types	183

<b>10.2.1</b>	<i>Defining union types</i>	183
<b>10.2.2</b>	<i>Restricting union types</i>	185
<b>10.2.3</b>	<i>Unions of unions</i>	186
<b>10.2.4</b>	<i>Specifying the member type in the instance</i>	187
<b>10.3</b>	<b>List types</b>	188
<b>10.3.1</b>	<i>Defining list types</i>	188
<b>10.3.2</b>	<i>Design hint: When should I use lists?</i>	189
<b>10.3.3</b>	<i>Restricting list types</i>	190
10.3.3.1	Length facets	192
10.3.3.2	Enumeration facet	192
10.3.3.3	Pattern facet	194
<b>10.3.4</b>	<i>Lists and strings</i>	195
<b>10.3.5</b>	<i>Lists of unions</i>	196
<b>10.3.6</b>	<i>Lists of lists</i>	196
<b>10.3.7</b>	<i>Restricting the item type</i>	198
	<b>Chapter 11 Built-in simple types</b>	<b>200</b>
<b>11.1</b>	<b>The XML Schema type system</b>	201
<b>11.1.1</b>	<i>The type hierarchy</i>	202
<b>11.1.2</b>	<i>Value spaces and lexical spaces</i>	204
<b>11.1.3</b>	<i>Facets and built-in types</i>	204
<b>11.2</b>	<b>String-based types</b>	205
<b>11.2.1</b>	<i>string, normalizedString, and token</i>	205
11.2.1.1	<i>Design hint: Should I use string, normalizedString, or token?</i>	207
<b>11.2.2</b>	<i>Name</i>	208
<b>11.2.3</b>	<i>NCName</i>	210
<b>11.2.4</b>	<i>language</i>	211
<b>11.3</b>	<b>Numeric types</b>	213

11.3.1	float <i>and</i> double	213
11.3.2	decimal	215
11.3.3	<i>Integer types</i>	217
11.3.3.1	Design hint: Is it an integer or a string?	220
<b>11.4</b>	<b>  Date and time types</b>	221
11.4.1	date	221
11.4.2	time	222
11.4.3	dateTime	223
11.4.4	dateTimeStamp	224
11.4.5	gYear	225
11.4.6	gYearMonth	226
11.4.7	gMonth	227
11.4.8	gMonthDay	227
11.4.9	gDay	228
11.4.10	duration	229
11.4.11	yearMonthDuration	231
11.4.12	dayTimeDuration	232
11.4.13	<i>Representing time zones</i>	233
11.4.14	<i>Facets</i>	234
11.4.15	<i>Date and time ordering</i>	235
<b>11.5</b>	<b>  Legacy types</b>	236
11.5.1	ID	236
11.5.2	IDREF	237
11.5.3	IDREFS	239
11.5.4	ENTITY	240
11.5.5	ENTITIES	242
11.5.6	NMTOKEN	243

11.5.7	NMTOKENS	244
11.5.8	NOTATION	245
11.6	Other types	246
11.6.1	QName	246
11.6.2	boolean	247
11.6.3	<i>The binary types</i>	248
11.6.4	anyURI	250
11.7	Comparing typed values	253
	<b>Chapter 12 Complex types</b>	<b>256</b>
12.1	What are complex types?	257
12.2	Defining complex types	258
12.2.1	<i>Named complex types</i>	258
12.2.2	<i>Anonymous complex types</i>	260
12.2.3	<i>Complex type alternatives</i>	261
12.3	Content types	262
12.3.1	<i>Simple content</i>	262
12.3.2	<i>Element-only content</i>	264
12.3.3	<i>Mixed content</i>	264
12.3.4	<i>Empty content</i>	265
12.4	Using element declarations	266
12.4.1	<i>Local element declarations</i>	266
12.4.2	<i>Element references</i>	267
12.4.3	<i>Duplication of element names</i>	268
12.5	Using model groups	270
12.5.1	sequence <i>groups</i>	270
12.5.1.1	Design hint: Should I care about the order of elements?	272
12.5.2	choice <i>groups</i>	273

12.5.3	<i>Nesting of sequence and choice groups</i>	275
12.5.4	<i>all groups</i>	276
12.5.5	<i>Named model group references</i>	278
12.5.6	<i>Deterministic content models</i>	279
12.6	Using attribute declarations	281
12.6.1	<i>Local attribute declarations</i>	281
12.6.2	<i>Attribute references</i>	282
12.6.3	<i>Attribute group references</i>	284
12.6.4	<i>Default attributes</i>	284
12.7	Using wildcards	284
12.7.1	<i>Element wildcards</i>	285
12.7.1.1	Controlling the namespace of replacement elements	287
12.7.1.2	Controlling the strictness of validation	287
12.7.1.3	Negative wildcards	289
12.7.2	<i>Open content models</i>	292
12.7.2.1	Open content in a complex type	292
12.7.2.2	Default open content	295
12.7.3	<i>Attribute wildcards</i>	298
<b>Chapter 13 Deriving complex types</b>		<b>300</b>
13.1	Why derive types?	301
13.2	Restriction and extension	302
13.3	Simple content and complex content	303
13.3.1	<i>simpleContent elements</i>	303
13.3.2	<i>complexContent elements</i>	304
13.4	Complex type extensions	305
13.4.1	<i>Simple content extensions</i>	306
13.4.2	<i>Complex content extensions</i>	307
13.4.2.1	Extending choice groups	309
13.4.2.2	Extending all groups	310

13.4.2.3	Extending open content	311
<b>13.4.3</b>	<i>Mixed content extensions</i>	312
<b>13.4.4</b>	<i>Empty content extensions</i>	313
<b>13.4.5</b>	<i>Attribute extensions</i>	314
<b>13.4.6</b>	<i>Attribute wildcard extensions</i>	315
<b>13.5</b>	<b>  Complex type restrictions</b>	316
<b>13.5.1</b>	<i>Simple content restrictions</i>	317
<b>13.5.2</b>	<i>Complex content restrictions</i>	318
13.5.2.1	Eliminating meaningless groups	320
13.5.2.2	Restricting element declarations	321
13.5.2.3	Restricting wildcards	322
13.5.2.4	Restricting groups	324
13.5.2.5	Restricting open content	329
<b>13.5.3</b>	<i>Mixed content restrictions</i>	331
<b>13.5.4</b>	<i>Empty content restrictions</i>	332
<b>13.5.5</b>	<i>Attribute restrictions</i>	333
<b>13.5.6</b>	<i>Attribute wildcard restrictions</i>	335
<b>13.5.7</b>	<i>Restricting types from another namespace</i>	337
13.5.7.1	Using targetNamespace on element and attribute declarations	339
<b>13.6</b>	<b>  Type substitution</b>	341
<b>13.7</b>	<b>  Controlling type derivation and substitution</b>	343
<b>13.7.1</b>	<i>final: Preventing complex type derivation</i>	343
<b>13.7.2</b>	<i>block: Blocking substitution of derived types</i>	344
<b>13.7.3</b>	<i>Blocking type substitution in element declarations</i>	346
<b>13.7.4</b>	<i>abstract: Forcing derivation</i>	346
	<b>Chapter 14 Assertions</b>	<b>350</b>
<b>14.1</b>	<b>  Assertions</b>	351
<b>14.1.1</b>	<i>Assertions for simple types</i>	353
14.1.1.1	Using XPath 2.0 operators	355

14.1.1.2	Using XPath 2.0 functions	357
14.1.1.3	Types and assertions	359
14.1.1.4	Inheriting simple type assertions	362
14.1.1.5	Assertions on list types	363
<b>14.1.2</b>	<b><i>Assertions for complex types</i></b>	<b>365</b>
14.1.2.1	Path expressions	367
14.1.2.2	Conditional expressions	369
14.1.2.3	Assertions in derived complex types	370
<b>14.1.3</b>	<b><i>Assertions and namespaces</i></b>	<b>372</b>
14.1.3.1	Using <code>xpathDefaultNamespace</code>	373
<b>14.2</b>	<b>  Conditional type assignment</b>	<b>375</b>
14.2.1	<i>The alternative element</i>	376
14.2.2	<i>Specifying conditional type assignment</i>	377
14.2.3	<i>Using XPath in the test attribute</i>	378
14.2.4	<i>The error type</i>	380
14.2.5	<i>Conditional type assignment and namespaces</i>	381
14.2.6	<i>Using inherited attributes in conditional type assignment</i>	382
<b>Chapter 15</b>	<b>Named groups</b>	<b>384</b>
<b>15.1</b>	<b>  Why named groups?</b>	<b>385</b>
<b>15.2</b>	<b>  Named model groups</b>	<b>386</b>
15.2.1	<i>Defining named model groups</i>	386
15.2.2	<i>Referencing named model groups</i>	388
15.2.2.1	Group references	388
15.2.2.2	Referencing a named model group in a complex type	389
15.2.2.3	Using <code>all</code> in named model groups	391
15.2.2.4	Named model groups referencing named model groups	392
<b>15.3</b>	<b>  Attribute groups</b>	<b>392</b>
15.3.1	<i>Defining attribute groups</i>	393
15.3.2	<i>Referencing attribute groups</i>	395
15.3.2.1	Attribute group references	395

15.3.2.2	Referencing attribute groups in complex types	396
15.3.2.3	Duplicate attribute names	397
15.3.2.4	Duplicate attribute wildcard handling	398
15.3.2.5	Attribute groups referencing attribute groups	398
<b>15.3.3</b>	<i>The default attribute group</i>	399
<b>15.4</b>	Named groups and namespaces	401
<b>15.5</b>	Design hint: Named groups or complex type derivations?	403
	<b>Chapter 16 Substitution groups</b>	<b>406</b>
<b>16.1</b>	Why substitution groups?	407
<b>16.2</b>	The substitution group hierarchy	408
<b>16.3</b>	Declaring a substitution group	409
<b>16.4</b>	Type constraints for substitution groups	412
<b>16.5</b>	Members in multiple groups	413
<b>16.6</b>	Alternatives to substitution groups	414
<b>16.6.1</b>	<i>Reusable choice groups</i>	414
<b>16.6.2</b>	<i>Substituting a derived type in the instance</i>	415
<b>16.7</b>	Controlling substitution groups	418
<b>16.7.1</b>	<i>final: Preventing substitution group declarations</i>	418
<b>16.7.2</b>	<i>block: Blocking substitution in instances</i>	419
<b>16.7.3</b>	<i>abstract: Forcing substitution</i>	420
	<b>Chapter 17 Identity constraints</b>	<b>422</b>
<b>17.1</b>	Identity constraint categories	423
<b>17.2</b>	Design hint: Should I use ID/IDREF or key/keyref?	424
<b>17.3</b>	Structure of an identity constraint	424
<b>17.4</b>	Uniqueness constraints	426

<b>17.5   Key constraints</b>	428
<b>17.6   Key references</b>	430
17.6.1 <i>Key references and scope</i>	432
17.6.2 <i>Key references and type equality</i>	432
<b>17.7   Selectors and fields</b>	433
17.7.1 <i>Selectors</i>	433
17.7.2 <i>Fields</i>	434
<b>17.8   XPath subset for identity constraints</b>	435
<b>17.9   Identity constraints and namespaces</b>	439
17.9.1 <i>Using xpathDefaultNamespace</i>	441
<b>17.10   Referencing identity constraints</b>	442
<b>Chapter 18 Redefining and overriding schema components</b>	<b>446</b>
<b>18.1   Redefinition</b>	448
18.1.1 <i>Redefinition basics</i>	448
18.1.1.1 Include plus redefine	450
18.1.1.2 Redefine and namespaces	450
18.1.1.3 Pervasive impact	450
18.1.2 <i>The mechanics of redefinition</i>	451
18.1.3 <i>Redefining simple types</i>	452
18.1.4 <i>Redefining complex types</i>	453
18.1.5 <i>Redefining named model groups</i>	454
18.1.5.1 Defining a subset	454
18.1.5.2 Defining a superset	455
18.1.6 <i>Redefining attribute groups</i>	456
18.1.6.1 Defining a subset	457
18.1.6.2 Defining a superset	458
<b>18.2   Overrides</b>	459
18.2.1 <i>Override basics</i>	459

18.2.1.1	Include plus override	461
18.2.1.2	Override and namespaces	461
18.2.1.3	Pervasive impact	462
<b>18.2.2</b>	<i>The mechanics of overriding components</i>	462
<b>18.2.3</b>	<i>Overriding simple types</i>	464
<b>18.2.4</b>	<i>Overriding complex types</i>	465
<b>18.2.5</b>	<i>Overriding element and attribute declarations</i>	466
<b>18.2.6</b>	<i>Overriding named groups</i>	467
<b>18.3</b>	Risks of redefines and overrides	468
18.3.1	<i>Risks of redefining or overriding types</i>	468
18.3.2	<i>Risks of redefining or overriding named groups</i>	470
	<b>Chapter 19 Topics for DTD users</b>	<b>472</b>
<b>19.1</b>	Element declarations	473
19.1.1	<i>Simple types</i>	474
19.1.2	<i>Complex types with simple content</i>	475
19.1.3	<i>Complex types with complex content</i>	476
19.1.4	<i>Mixed content</i>	478
19.1.5	<i>Empty content</i>	479
19.1.6	<i>Any content</i>	480
<b>19.2</b>	Attribute declarations	480
19.2.1	<i>Attribute types</i>	480
19.2.2	<i>Enumerated attribute types</i>	481
19.2.3	<i>Notation attributes</i>	482
19.2.4	<i>Default values</i>	482
<b>19.3</b>	Parameter entities for reuse	483
19.3.1	<i>Reusing content models</i>	484
19.3.2	<i>Reusing attributes</i>	485
<b>19.4</b>	Parameter entities for extensibility	486

19.4.1	<i>Extensions for sequence groups</i>	486	■
19.4.2	<i>Extensions for choice groups</i>	489	■
19.4.3	<i>Attribute extensions</i>	490	■
19.5	External parameter entities	492	■
19.6	General entities	493	
19.6.1	<i>Character and other parsed entities</i>	493	
19.6.2	<i>Unparsed entities</i>	493	
19.7	Notations	493	
19.7.1	<i>Declaring a notation</i>	494	■
19.7.2	<i>Declaring a notation attribute</i>	495	■
19.7.3	<i>Notations and unparsed entities</i>	496	
19.8	Comments	497	
19.9	Using DTDs and schemas together	499	
	<b>Chapter 20 XML information modeling</b>	<b>500</b>	
20.1	Data modeling paradigms	502	
20.2	Relational models	503	
20.2.1	<i>Entities and attributes</i>	504	
20.2.2	<i>Relationships</i>	507	
20.2.2.1	One-to-one and one-to-many relationships	507	
20.2.2.2	Many-to-many relationships	507	
20.2.2.2.1	<i>Approach #1: Use containment with repetition</i>	508	
20.2.2.2.2	<i>Approach #2: Use containment with references</i>	510	
20.2.2.2.3	<i>Approach #3: Use relationship elements</i>	512	
20.3	Modeling object-oriented concepts	514	
20.3.1	<i>Inheritance</i>	514	
20.3.2	<i>Composition</i>	519	
20.4	Modeling web services	522	

<b>20.5</b>	<b>  Considerations for narrative content</b>	524
<b>20.5.1</b>	<i>Semantics vs. style</i>	524
20.5.1.1	Benefits of excluding styling	524
20.5.1.2	Rendition elements: “block” and “inline”	525
<b>20.5.2</b>	<i>Considerations for schema design</i>	526
20.5.2.1	Flexibility	526
20.5.2.2	Reusing existing vocabularies	526
20.5.2.3	Attributes are for metadata	526
20.5.2.4	Humans write the documents	527
<b>20.6</b>	<b>  Considerations for a hierarchical model</b>	527
<b>20.6.1</b>	<i>Intermediate elements</i>	527
<b>20.6.2</b>	<i>Wrapper lists</i>	531
<b>20.6.3</b>	<i>Level of granularity</i>	532
<b>20.6.4</b>	<i>Generic vs. specific elements</i>	533
		■
		■
		■
		■
		■
<b>Chapter 21</b>	<b>Schema design and documentation</b>	<b>538</b>
<b>21.1</b>	<b>  The importance of schema design</b>	539
<b>21.2</b>	<b>  Uses for schemas</b>	540
<b>21.3</b>	<b>  Schema design goals</b>	542
<b>21.3.1</b>	<i>Flexibility and extensibility</i>	542
<b>21.3.2</b>	<i>Reusability</i>	543
<b>21.3.3</b>	<i>Clarity and simplicity</i>	545
21.3.3.1	Naming and documentation	545
21.3.3.2	Clarity of structure	546
21.3.3.3	Simplicity	546
<b>21.3.4</b>	<i>Support for graceful versioning</i>	547
<b>21.3.5</b>	<i>Interoperability and tool compatibility</i>	547
<b>21.4</b>	<b>  Developing a schema design strategy</b>	548
<b>21.5</b>	<b>  Schema organization considerations</b>	550
<b>21.5.1</b>	<i>Global vs. local components</i>	550

21.5.1.1	Russian Doll	551
21.5.1.2	Salami Slice	553
21.5.1.3	Venetian Blind	554
21.5.1.4	Garden of Eden	555
<b>21.5.2</b>	<b><i>Modularizing schema documents</i></b>	<b>557</b>
<b>21.6</b>	<b>  Naming considerations</b>	<b>559</b>
21.6.1	<i>Rules for valid XML names</i>	559
21.6.2	<i>Separators</i>	560
21.6.3	<i>Name length</i>	560
21.6.4	<i>Standard terms and abbreviations</i>	561
21.6.5	<i>Use of object terms</i>	562
<b>21.7</b>	<b>  Namespace considerations</b>	<b>564</b>
21.7.1	<i>Whether to use namespaces</i>	564
21.7.2	<i>Organizing namespaces</i>	565
21.7.2.1	Same namespace	565
21.7.2.2	Different namespaces	568
21.7.2.3	Chameleon namespaces	572
21.7.3	<i>Qualified vs. unqualified forms</i>	575
21.7.3.1	Qualified local names	575
21.7.3.2	Unqualified local names	576
21.7.3.3	Using form in schemas	576
21.7.3.4	Form and global element declarations	578
21.7.3.5	Default namespaces and unqualified names	578
21.7.3.6	Qualified vs. unqualified element names	579
21.7.3.7	Qualified vs. unqualified attribute names	580
<b>21.8</b>	<b>  Schema documentation</b>	<b>580</b>
21.8.1	<i>Annotations</i>	581
21.8.2	<i>User documentation</i>	582
21.8.2.1	Documentation syntax	582
21.8.2.2	Data element definitions	584
21.8.2.3	Code documentation	585
21.8.2.4	Section comments	585

<b>21.8.3</b>	<i>Application information</i>	586
<b>21.8.4</b>	<i>Non-native attributes</i>	588
21.8.4.1	Design hint: Should I use annotations or non-native attributes?	589
<b>21.8.5</b>	<i>Documenting namespaces</i>	589
	<b>Chapter 22 Extensibility and reuse</b>	<b>594</b>
<b>22.1</b>	<b>Reuse</b>	596
<b>22.1.1</b>	<i>Reusing schema components</i>	596
<b>22.1.2</b>	<i>Creating schemas that are highly reusable</i>	597
<b>22.1.3</b>	<i>Developing a common components library</i>	597
<b>22.2</b>	<b>Extending schemas</b>	599
<b>22.2.1</b>	<i>Wildcards</i>	601
<b>22.2.2</b>	<i>Open content</i>	604
<b>22.2.3</b>	<i>Type substitution</i>	605
<b>22.2.4</b>	<i>Substitution groups</i>	607
<b>22.2.5</b>	<i>Type redefinition</i>	609
<b>22.2.6</b>	<i>Named group redefinition</i>	611
<b>22.2.7</b>	<i>Overrides</i>	612
	<b>Chapter 23 Versioning</b>	<b>616</b>
<b>23.1</b>	<b>Schema compatibility</b>	617
<b>23.1.1</b>	<i>Backward compatibility</i>	618
<b>23.1.2</b>	<i>Forward compatibility</i>	623
<b>23.2</b>	<b>Using version numbers</b>	626
<b>23.2.1</b>	<i>Major and minor versions</i>	626
<b>23.2.2</b>	<i>Placement of version numbers</i>	628
23.2.2.1	Version numbers in schema documents	628
23.2.2.2	Versions in schema locations	630
23.2.2.3	Versions in instances	631
23.2.2.4	Versions in namespace names	632
23.2.2.5	A combination strategy	633

<b>23.3</b>	Application compatibility	634
<b>23.4</b>	Lessening the impact of versioning	635
<b>23.4.1</b>	<i>Define a versioning strategy</i>	636
<b>23.4.2</b>	<i>Make only necessary changes</i>	636
<b>23.4.3</b>	<i>Document all changes</i>	637
<b>23.4.4</b>	<i>Deprecate components before deleting them</i>	638
<b>23.4.5</b>	<i>Provide a conversion capability</i>	639
<b>23.5</b>	Versions of the XML Schema language	639
<b>23.5.1</b>	<i>New features in version 1.1</i>	640
<b>23.5.2</b>	<i>Forward compatibility of XML Schema 1.1</i>	641
<b>23.5.3</b>	<i>Portability of implementation-defined types and facets</i>	642
23.5.3.1	Using <code>typeAvailable</code> and <code>typeUnavailable</code>	644
23.5.3.2	Using <code>facetAvailable</code> and <code>facetUnavailable</code>	645
<b>Appendix A</b>	<b>XSD keywords</b>	<b>648</b>
<b>A.1</b>	Elements	649
<b>A.2</b>	Attributes	671
<b>Appendix B</b>	<b>Built-in simple types</b>	<b>690</b>
<b>B.1</b>	Built-in simple types	691
<b>B.2</b>	Applicability of facets to built-in simple types	695
	<b>Index</b>	<b>699</b>



# Foreword

**clas·sic** (*adjective*)

judged over a period of time to be important and of the highest quality:

*a classic novel*

*a classic car*

Neither this definition, nor any of the leading dictionary definitions, has a usage example anything like:

*a classic work on high-tech software*

After all, it is a rare book on software that even survives long enough to be “judged over a period of time.”

Nevertheless, *Definitive XML Schema* satisfies every definition of “classic.” It is one of the elite few software books that have been in print continuously for over ten years, and an essential trustworthy guide for tens of thousands of readers.

This *Second Edition* continues to be an essential and trustworthy classic:

**Essential** because in the last ten years XML has become the accepted standard for data interchange, and XML Schema 1.0 is largely responsible. Now version 1.1 has extended the ability to specify and

validate document data, to a degree previously possible only for databases. These updates are covered in this book by extensive revisions—the most significant 250 of which are flagged in the text and table of contents. Hundreds more unflagged revisions reflect W3C corrections of XML Schema errata, and ten years of evolving “best practices.”

**Trustworthy** because it is both authoritative and accurate.

- The **author(ity)**, Priscilla Walmsley, is a noted consultant who has been using XML Schema ever since she helped develop it as a member of the W3C XML Schema Group. She personally devised many of the current “best practices” described in this book. Priscilla is the Editor of the W3C XML Schema Primer, Second Edition.
- **Accuracy** was preserved by using the same XML-based production system that was used in 2002, operated by the same team of XML experts who read and thoroughly understood the book. Priscilla’s original XML source (in which she had personally tagged the version 1.1 revisions) was used throughout production. Dmitry Kirsanov copy-edited and proofed it, while Alina Kirsanova prepared the index, coded the XSL transformations, and generated the camera-ready PDFs.

The result, as you will see, retains the structure, clarity, patient explanations, validated examples (over 450!), and well-reasoned advice that critics praised in the 2002 edition—but now they are ten years more up-to-date.

And after you’ve read *Definitive XML Schema, Second Edition*, it won’t take another ten years for you, too, to judge it a classic.

*Charles F. Goldfarb*  
*Belmont, CA*  
*August 2012*

# Acknowledgments

First and foremost, I would like to thank Charles Goldfarb for his invaluable guidance and support. Alina Kirsanova and Dmitry Kirsanov did an excellent job preparing this book for publication. I would also like to thank Mark Taub at Prentice Hall for his hand in the making this work possible.

Of course, this book would not have been possible without the efforts of all of the members of the W3C XML Schema Working Group, with whom I have had the pleasure of working for six years. The content of this book was shaped by the questions and comments of the people who contribute to XML-DEV and xmlschema-dev.

Finally, I'd like to thank my Dad for teaching me to “get stuck into it,” a skill which allowed me to complete this substantial project.

*Priscilla Walmsley  
Traverse City, Michigan  
March 2012*



# How to use this book

This book covers the two versions of XML Schema—1.0 and 1.1—and provides revision bars to assist access to just one or the other. In referring to both versions as “XML Schema,” the book follows customary practice, despite the official name of 1.1 being “W3C XML Schema Definition Language (XSD) 1.1.” For either version, the book is useable as both a tutorial and a reference.

As a tutorial, the book can be read from cover to cover with confidence that each topic builds logically on the information that was previously covered. (Of course, knowledge of XML itself is always a prerequisite to learning about XML Schema, and is assumed in this book.)

When using this book as a reference, you have several access options available to you:

- A comprehensive index starts on page 699.
- An alphabetical list of all the elements and attributes that make up the XML Schema syntax is in Appendix A on p. 648. For each there is a reference to further coverage in the body of the book.
- XML Schema includes a basic set of datatypes, known formally as the “built-in simple types.” They are listed in Appendix B on p. 690. This appendix also refers to more detailed descriptions in the body of the book.

- The major changes in version 1.1 of XML Schema are summarized in Section 23.5.1 on p. 640, with references to detailed coverage elsewhere in the book.

## Revisions in the Second Edition

This edition of *Definitive XML Schema* contains more than 500 revisions, covering such new and updated topics as:

- W3C published corrections for errata in XML Schema 1.0
- Current “best practices” after ten years of experience
- XML information modeling for relational and object-oriented modeling paradigms
- Schema design: evaluating pros and cons of alternatives
- Schema strategy: formulating a coherent strategy for schema development and maintenance
- Version 1.1 updates and additions to XML Schema

### *Identifying 1.1-related revisions*

The author has chosen a “1.1 subset” of the book revisions, comprising the 250 most significant revisions that deal with version 1.1. If a section, table, example, or paragraph has content entirely from the “1.1 subset,” there is a solid gray revision bar on its right. If other material might be included, the bar is a gray dotted line.

### *Strategies for using the revision bars*

If your interest is **solely 1.0** (perhaps because your software does not yet support 1.1), you may decide to focus on content that either has a dotted revision bar or no bar at all.

If you are interested only in what is **new in 1.1** (presumably because you already know 1.0), consider content having either a solid or dotted revision bar in deciding where to focus your reading.

Finally, if your interest is **all of 1.1** (because you don't already know 1.0), you can easily disregard the revision bars (that's why they are grayed out ☺).

## Syntax tables

This book contains syntax tables, each summarizing the allowed syntax of an XML Schema component. The first such table does not occur until Section 4.2 on p. 58, by which point the undefined terms in this explanation will have been introduced.

Syntax tables, whose captions all start with “XSD Syntax,” look like the example below, which shows the syntax for named simple types. It contains the following information:

- The element name(s) used for this XML Schema component.
- The possible parent element name(s). Note that “1.1”, printed white on a gray box, precedes `override` to identify it as a construct that is only permitted in version 1.1. This convention is followed in all syntax tables; it occurs once more in this table.
- A list of allowed attributes, along with their types, valid values, and brief descriptions. The names of required attributes appear in bold font. Default values appear in italics in the *Type* column.
- The allowed child elements, shown as a content model that uses, for compactness, the XML DTD syntax. Commas indicate that child elements must appear in the order shown, while vertical bars (|) indicate a choice among child elements. Occurrence constraints indicate how many of each may appear: ? means zero or one, \* means zero or more, and + means one or more. Otherwise, one and only one is required. In this example, the allowed content is zero or one `annotation` element, followed by a choice of either one `restriction`, one `list`, or one `union` element.

**Table** XSD Syntax: named simple type definition

---

<i>Name</i>		
simpleType		
<hr/>		
<i>Parents</i>		
schema, redefine,  override		
<hr/>		
<i>Attribute name</i>	<i>Type</i>	<i>Description</i>
id	ID	Unique ID.
<b>name</b>	NCName	Simple type name.
final	"#all"   list of ("restriction"   "list"   "union"    "extension")	Whether other types can be derived from this one.
<hr/>		
<i>Content</i>		
annotation?, (restriction   list   union)		
<hr/>		

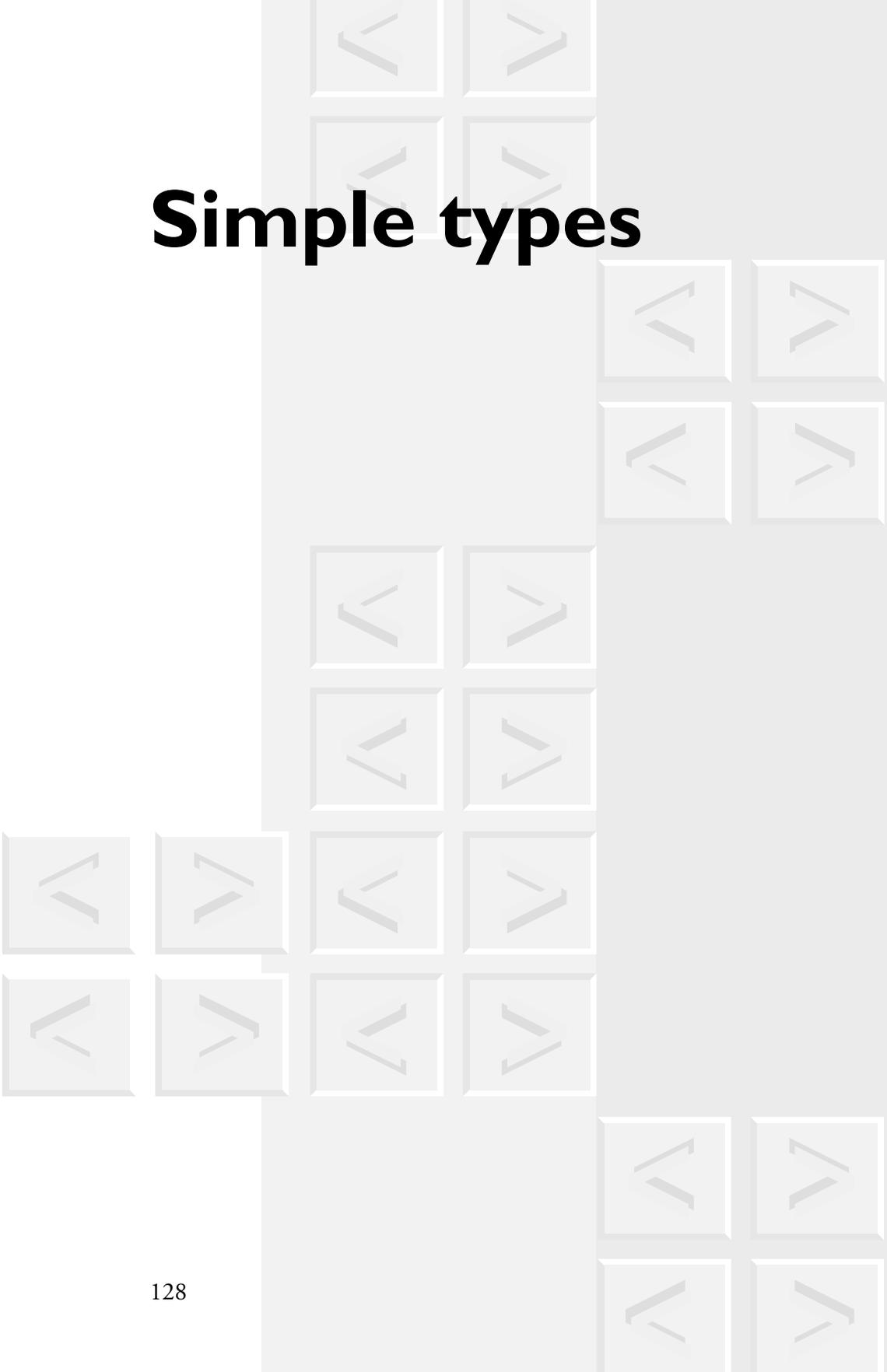
In some cases, there is more than one syntax table for the same element name, because certain element names in XML Schema have multiple uses. For example, `simpleType` is used for both named simple types and anonymous simple types. Each of these use cases of `simpleType` allows different attributes and a different set of parent elements, so each is described with its own table.

## Companion website

This book has a companion website, maintained by the author, at [www.datypic.com/books/defxmschema2](http://www.datypic.com/books/defxmschema2). On the website, you can view any errata and download the examples from this book. In addition to the examples that appear in the book, which are generally concise in order to illustrate a particular point, the website also has larger, more comprehensive instances and schemas that can be copied or used to test validation.



# Simple types



# Chapter 8

**B**

oth element and attribute declarations can use simple types to describe their data content. This chapter introduces simple types and explains how to define your own atomic simple types for use in your schemas.

## 8.1 | Simple type varieties

There are three varieties of simple types: atomic types, list types, and union types.

1. *Atomic types* have values that are indivisible, such as 10 or large.
2. *List types* have values that are whitespace-separated lists of atomic values, such as `<availableSizes>10 large 2</availableSizes>`.
3. *Union types* may have values that are either atomic values or list values. What differentiates them is that the set of valid values, or “value space,” for the type is the union of the value spaces of

two or more other simple types. For example, to represent a dress size, you may define a union type that allows a value to be either an integer from 2 through 18, or one of the string values `small`, `medium`, or `large`.

List and union types are covered in Chapter 10.

### 8.1.1 *Design hint: How much should I break down my data values?*

Data values should be broken down to the most atomic level possible. This allows them to be processed in a variety of ways for different uses, such as display, mathematical operations, and validation. It is much easier to concatenate two data values back together than it is to split them apart. In addition, more granular data is easier to validate.

It is a fairly common practice to put a data value and its units in the same element, for example `<length>3cm</length>`. However, the preferred approach is to have a separate data value, preferably an attribute, for the units, for example `<length units="cm">3</length>`.

Using a single concatenated value is limiting because:

- It is extremely cumbersome to validate. You have to apply a complicated pattern that would need to change every time a unit type is added.
- You cannot perform comparisons, conversions, or mathematical operations on the data without splitting it apart.
- If you want to display the data item differently (for example, as “3 centimeters” or “3 cm” or just “3”, you have to split it apart. This complicates the stylesheets and applications that process instance documents.

It is possible to go too far, though. For example, you may break a date down as follows:

```
<orderDate>  
  <year>2001</year>  
  <month>06</month>  
  <day>15</day>  
</orderDate>
```

This is probably overkill unless you have a special need to process these items separately.

## 8.2 | Simple type definitions

### 8.2.1 *Named simple types*

Simple types can be either named or anonymous. Named simple types are always defined globally (i.e., their parent is always `schema`<sup>1</sup>) and are required to have a name that is unique among the types (both simple and complex) in the schema. The syntax for a named simple type definition is shown in Table 8–1.

The name of a simple type must be an XML non-colonized name, which means that it must start with a letter or underscore, and may only contain letters, digits, underscores, hyphens, and periods. You cannot include a namespace prefix when defining the type; it takes its namespace from the target namespace of the schema document.

All examples of named types in this book have the word “Type” at the end of their names to clearly distinguish them from element and attribute names. However, this is a convention and not a requirement. You can even have a type definition and an element declaration using the same name, but this is not recommended because it can be confusing.

Example 8–1 shows the definition of a named simple type `DressSizeType` along with an element declaration that references it. Named types can be used in multiple element and attribute declarations.

---

1. Except in the case of a redefine or override.

**Table 8–1** XSD Syntax: named simple type definition

<i>Name</i>		
simpleType		
<i>Parents</i>		
schema, redefine,  override		
<i>Attribute name</i>	<i>Type</i>	<i>Description</i>
id	ID	Unique ID.
<b>name</b>	NCName	Simple type name.
final	"#all"   list of ("restriction"   "list"   "union"    "extension")	Whether other types can be derived from this one (see Section 8.5); defaults to finalDefault of schema.
<i>Content</i>		
annotation?, (restriction   list   union)		

**Example 8–1.** Defining and referencing a named simple type

```
<xs:simpleType name="DressSizeType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="2"/>
    <xs:maxInclusive value="18"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="size" type="DressSizeType"/>
```

**8.2.2** *Anonymous simple types*

Anonymous types, on the other hand, must not have names. They are always defined entirely within an element or attribute declaration, and may only be used once, by that declaration. Defining a type anonymously prevents it from ever being restricted, used in a list or

union, redefined, or overridden. The syntax to define an anonymous simple type is shown in Table 8–2.

**Table 8–2** XSD Syntax: anonymous simple type definition

<i>Name</i>		
simpleType		
<i>Parents</i>		
element, attribute, restriction, list, union,  alternative		
<i>Attribute name</i>	<i>Type</i>	<i>Description</i>
id	ID	Unique ID.
<i>Content</i>		
annotation?, (restriction   list   union)		

Example 8–2 shows the definition of an anonymous simple type within an element declaration.

### **Example 8–2.** Defining an anonymous simple type

```
<xs:element name="size">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="2"/>
      <xs:maxInclusive value="18"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

### **8.2.3** *Design hint: Should I use named or anonymous types?*

The advantage of named types is that they may be defined once and used many times. For example, you may define a type named

`ProductCodeType` that lists all of the valid product codes in your organization. This type can then be used in many element and attribute declarations in many schemas. This has the advantages of

- Encouraging consistency throughout the organization
- Reducing the possibility of error
- Requiring less time to define new schemas
- Simplifying maintenance, because new product codes need only be added in one place

If a type is named, you can also derive new types from it, which is another way to promote reuse and consistency.

Named types can also make a schema more readable when its type definitions are complicated.

An anonymous type, on the other hand, can be used only in the element or attribute declaration that contains it. It can never be redefined, overridden, have types derived from it, or be used in a list or union type. This can seriously limit its reusability, extensibility, and ability to change over time.

However, there are cases where anonymous types are preferable to named types. If the type is unlikely to ever be reused, the advantages listed above no longer apply. Also, there is such a thing as too much reuse. For example, if an element can contain the values 1 through 10, it does not make sense to define a type named `OneToTenType` to be reused by other unrelated element declarations with the same value space. If the value space for one of the element declarations using that named type changes but the other element declarations stay the same, it actually makes maintenance more difficult, because a new type would need to be defined at that time.

In addition, anonymous types can be more readable when they are relatively simple. It is sometimes desirable to have the definition of the type right there with the element or attribute declaration.

## 8.3 | Simple type restrictions

Every simple type is a restriction of another simple type, known as its base type. It is not possible to extend a simple type, except by adding attributes which results in a complex type. This is described in Section 13.4.1 on p. 306.

Every new simple type restricts the value space of its base type in some way. Example 8–3 shows a definition of `DressSizeType` that restricts the built-in type `integer`.

---

### *Example 8–3.* Deriving a simple type from a built-in simple type

---

```
<xs:simpleType name="DressSizeType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="2"/>
    <xs:maxInclusive value="18"/>
    <xs:pattern value="\d{1,2}"/>
  </xs:restriction>
</xs:simpleType>
```

---

Simple types may also restrict user-derived simple types that are defined in the same schema document, or even in a different schema document. For example, you could further restrict `DressSizeType` by defining another simple type, `MediumDressSizeType`, as shown in Example 8–4.

A simple type restricts its base type by applying facets to restrict its values. In Example 8–4, the facets `minInclusive` and `maxInclusive` are used to restrict the value of `MediumDressSizeType` to be between 8 and 12 inclusive.

---

### *Example 8–4.* Deriving a simple type from a user-derived simple type

---

```
<xs:simpleType name="MediumDressSizeType">
  <xs:restriction base="DressSizeType">
    <xs:minInclusive value="8"/>
    <xs:maxInclusive value="12"/>
  </xs:restriction>
</xs:simpleType>
```

---

### 8.3.1 Defining a restriction

The syntax for a restriction element is shown in Table 8–3. You must specify one base type either by using the `base` attribute or by defining the simple type anonymously using a `simpleType` child. The option of using a `simpleType` child is generally only useful when restricting list types, as described in Section 10.3.3 on p. 190.

**Table 8–3** XSD Syntax: simple type restriction

<i>Name</i>		
restriction		
<i>Parents</i>		
simpleType		
<i>Attribute name</i>	<i>Type</i>	<i>Description</i>
id	ID	Unique ID.
base	QName	Simple type that is being restricted; either a base attribute or a <code>simpleType</code> child is required.
<i>Content</i>		
annotation?, simpleType?, (minExclusive   minInclusive             maxExclusive   maxInclusive   length   minLength   maxLength             totalDigits   fractionDigits   enumeration   pattern             whitespace    assertion    explicitTimezone    {any element in another namespace})*		

Within a restriction element, you can specify any of the facets, in any order. However, the only facets that may appear more than once in the same restriction are `pattern`, `enumeration`, and `assertion`. It is legal to define a restriction that has no facets specified. In this case, the derived type allows the same values as the base type.

### 8.3.2 Overview of the facets

The available facets are listed in Table 8–4.

**Table 8–4** Facets

<i>Facet</i>	<i>Meaning</i>
<code>minExclusive</code>	Value must be greater than $x$ .
<code>minInclusive</code>	Value must be greater than or equal to $x$ .
<code>maxInclusive</code>	Value must be less than or equal to $x$ .
<code>maxExclusive</code>	Value must be less than $x$ .
<code>length</code>	The length of the value must be equal to $x$ .
<code>minLength</code>	The length of the value must be greater than or equal to $x$ .
<code>maxLength</code>	The length of the value must be less than or equal to $x$ .
<code>totalDigits</code>	The number of significant digits must be less than or equal to $x$ .
<code>fractionDigits</code>	The number of fractional digits must be less than or equal to $x$ .
<code>whiteSpace</code>	The schema processor should either preserve, replace, or collapse whitespace depending on $x$ .
<code>enumeration</code>	$x$ is one of the valid values.
<code>pattern</code>	$x$ is one of the regular expressions that the value may match.
<code>explicitTimezone</code>	The time zone part of the date/time value is required, optional, or prohibited depending on $x$ .
<code>assertion</code>	The value must conform to a constraint in the XPath expression.

The syntax for applying a facet is shown in Table 8–5. All facets (except `assertion`) must have a `value` attribute, which has different

valid values depending on the facet. Most facets may also have a `fixed` attribute, as described in Section 8.3.4 on p. 140.

**Table 8–5** XSD Syntax: facet

<i>Name</i>		
minExclusive, minInclusive, maxExclusive, maxInclusive, length, minLength, maxLength, totalDigits, fractionDigits, enumeration, pattern, whiteSpace,  explicitTimezone <sup>†</sup>		
<i>Parents</i>		
restriction		
<i>Attribute name</i>	<i>Type</i>	<i>Description</i>
id	ID	Unique ID.
<b>value</b>	various	Value of the restricting facet.
fixed	boolean: <i>false</i>	Whether the facet is fixed and therefore cannot be restricted further (see Section 8.3.4); not applicable for pattern, enumeration.
<i>Content</i>		
annotation?		
† The assertion facet has a different syntax that is described in Table 14–1.		

Certain facets are not applicable to some types. For example, it does not make sense to apply the `fractionDigits` facet to a character string type. There is a defined set of applicable facets for each of the built-in types.<sup>1</sup> If a facet is applicable to a built-in type, it is also applicable to atomic types that are derived from it. For example, since the `length` facet is applicable to `string`, if you derive a new type from

1. Technically, it is the primitive types that have applicable facets, with the rest of the built-in types inheriting that applicability from their base types. However, since most people do not have the built-in type hierarchy memorized, it is easier to list applicable facets for all the built-in types.

string, the length facet is also applicable to your new type. Section 8.4 on p. 142 describes each of the facets in detail and lists the built-in types to which the facet can apply.

### 8.3.3 *Inheriting and restricting facets*

When a simple type restricts its base type, it inherits all of the facets of its base type, its base type's base type, and so on back through its ancestors. Example 8–4 showed a simple type `MediumDressSizeType` whose base type is `DressSizeType`. `DressSizeType` has a pattern facet which restricts its value space to one- or two-digit numbers. Since `MediumDressSizeType` inherits all of the facets from `DressSizeType`, this same pattern facet applies to `MediumDressSizeType` also. Example 8–5 shows an equivalent definition of `MediumDressSizeType` where it restricts `integer` and has the pattern facet applied.

#### **Example 8–5.** Effective definition of `MediumDressSizeType`

---

```
<xs:simpleType name="MediumDressSizeType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="8"/>
    <xs:maxInclusive value="12"/>
    <xs:pattern value="\d{1,2}"/>
  </xs:restriction>
</xs:simpleType>
```

---

Sometimes a simple type definition will include facets that are also specified for one of its ancestors. In Example 8–4, `MediumDressSizeType` includes `minInclusive` and `maxInclusive`, which are also applied to its base type, `DressSizeType`. The `minInclusive` and `maxInclusive` facets of `MediumDressSizeType` (whose values are 8 and 12, respectively) override those of `DressSizeType` (2 and 18, respectively).

It is a requirement that the facets of a derived type (in this case `MediumDressSizeType`) be more restrictive than those of the base type. In Example 8–6, we define a new restriction of `DressSizeType`,

called `SmallDressSizeType`, and set `minInclusive` to 0. This type definition is illegal, because it attempts to expand the value space by allowing 0, which was not valid for `DressSizeType`.

**Example 8–6.** Illegal attempt to extend a simple type

---

```
<xs:simpleType name="SmallDressSizeType">
  <xs:restriction base="DressSizeType">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="6"/>
  </xs:restriction>
</xs:simpleType>
```

---

This rule also applies when you are restricting the built-in types. For example, the `short` type has a `maxInclusive` value of 32767. It is illegal to define a restriction of `short` that sets `maxInclusive` to 32768.

Although enumeration facets can appear multiple times in the same type definition, they are treated in much the same way. If both a derived type and its ancestor have a set of enumeration facets, the values of the derived type must be a subset of the values of the ancestor. An example of this is provided in Section 8.4.4 on p. 145.

Likewise, the pattern facets specified in a derived type must allow a subset of the values allowed by the ancestor types. A schema processor will not necessarily check that the regular expressions represent a subset; instead, it will validate instances against the patterns of both the derived type and all the ancestor types, effectively taking the intersection of the pattern values.

### 8.3.4 *Fixed facets*

When you define a simple type, you can fix one or more of the facets. This means that further restrictions of this type cannot change the value of the facet. Any of the facets may be fixed, with the exception of `pattern`, `enumeration`, and `assertion`. Example 8–7 shows our

`DressSizeType` with fixed `minInclusive` and `maxInclusive` facets, as indicated by a fixed attribute set to `true`.

### **Example 8–7. Fixed facets**

---

```
<xs:simpleType name="DressSizeType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="2" fixed="true"/>
    <xs:maxInclusive value="18" fixed="true"/>
    <xs:pattern value="\d{1,2}"/>
  </xs:restriction>
</xs:simpleType>
```

---

With this definition of `DressSizeType`, it would have been illegal to define the `MediumDressSizeType` as shown in Example 8–4 because it attempts to override the `minInclusive` and `maxInclusive` facets which are now fixed. Some of the built-in types have fixed facets that cannot be overridden. For example, the built-in type `integer` has its `fractionDigits` facet fixed at 0, so it is illegal to derive a type from `integer` and specify a `fractionDigits` that is not 0.

#### **8.3.4.1** Design hint: When should I fix a facet?

Fixing facets makes your type less flexible and discourages other schema authors from reusing it. Keep in mind that any types that may be derived from your type must be more restrictive, so you are not at risk that your type will be dramatically changed if its facets are unfixed.

A justification for fixing facets might be that changing that facet value would significantly alter the meaning of the type. For example, suppose you want to define a simple type that represents price. You define a `Price` type and fix the `fractionDigits` at 2. This still allows other schema authors to restrict `Price` to define other types, for example, by limiting it to a certain range of values. However, they cannot modify the `fractionDigits` of the type, because this would result in a type not representing a price in dollars and cents.

## 8.4 | Facets

### 8.4.1 *Bounds facets*

The four bounds facets (`minInclusive`, `maxInclusive`, `minExclusive`, and `maxExclusive`) restrict a value to a specified range. Our previous examples applied `minInclusive` and `maxInclusive` to restrict the value space of `DressSizeType`. While `minInclusive` and `maxInclusive` specify boundary values that are included in the valid range, `minExclusive` and `maxExclusive` specify bounds that are excluded from the valid range.

There are several constraints associated with the bounds facets:

- `minInclusive` and `minExclusive` cannot both be applied to the same type. Likewise, `maxInclusive` and `maxExclusive` cannot both be applied to the same type. You may, however, mix and match, applying, for example, `minInclusive` and `maxExclusive` together. You may also apply just one end of the range, such as `minInclusive` only.
- The value for the lower bound (`minInclusive` or `minExclusive`) must be less than or equal to the value for the upper bound (`maxInclusive` or `maxExclusive`).
- The facet value must be a valid value for the base type. For example, when restricting `integer`, it is illegal to specify a `maxInclusive` value of 18.5, because 18.5 is not a valid `integer`.

The four bounds facets can be applied only to the date/time and numeric types, and the types derived from them. Special consideration should be given to time zones when applying bounds facets to date/time types. For more information, see Section 11.4.15 on p. 235.

## 8.4.2 *Length facets*

The `length` facet allows you to limit values to a specific length. If it is a string-based type, length is measured in number of characters. This includes the XML DTD types and `anyURI`. If it is a binary type, length is measured in octets of binary data. If it is a list type, length is measured as the number of items in the list. The facet value for `length` must be a nonnegative integer.

The `minLength` and `maxLength` facets allow you to limit a value's length to a specific range. Either of both of these facets may be applied. If they are both applied, `minLength` must be less than or equal to `maxLength`. If the `length` facet is applied, neither `minLength` nor `maxLength` may be applied. The facet values for `minLength` and `maxLength` must be nonnegative integers.

The three length facets (`length`, `minLength`, `maxLength`) can be applied to any string-based types (including the XML DTD types), the binary types, and `anyURI`. They cannot be applied to the date/time types, numeric types, or `boolean`.

### 8.4.2.1 Design hint: What if I want to allow empty values?

Many of the built-in types do not allow empty values. Types other than `string`, `normalizedString`, `token`, `hexBinary`, `base64Binary`, and `anyURI` do not allow empty values unless `xsi:nil` appears in the element tag.

You may have an integer that you want to be either between 2 and 18, or empty. First, consider whether you want to make the element (or attribute) optional. In this case, if the data is absent, the element will not appear at all. However, sometimes it is desirable for the element to appear, as a placeholder, or perhaps it is unavoidable because of the technology used to generate the instance.

If you do determine that the elements must be able to appear empty, you must define a union type that includes both the integer type and an empty string, as shown in Example 8–8.

**Example 8–8.** Union allowing an empty value

---

```
<xs:simpleType name="DressSizeType">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="2"/>
        <xs:maxInclusive value="18"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value=""/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

---

**8.4.2.2** Design hint: What if I want to restrict the length of an integer?

The length facet only applies to the string-based types, the XML DTD types, the binary types, and anyURI. It does not make sense to try to limit the length of the date/time types because they have fixed lexical representations. But what if you want to restrict the length of an integer value?

You can restrict the lower and upper bounds of an integer by applying bounds facets, as discussed in Section 8.4.1 on p. 142. You can also control the number of significant digits in an integer using the totalDigits facet, as discussed in Section 8.4.3 on p. 145. However, these facets do not consider leading zeros as significant. Therefore, they cannot force an integer to appear in the instance with a specific number of digits. To do this, you need a pattern. For example, the pattern `\d{1,2}` used in our DressSizeType example forces the size to be one or two digits long, so `012` would be invalid.

Before taking this approach, however, you should reconsider whether it is really an integer or a string. See Section 11.3.3.1 on p. 220 for a discussion of this issue.

### 8.4.3 totalDigits *and* fractionDigits

The `totalDigits` facet allows you to specify the maximum number of digits in a number. The facet value for `totalDigits` must be a positive integer.

The `fractionDigits` facet allows you to specify the maximum number of digits in the fractional part of a number. The facet value for `fractionDigits` must be a nonnegative integer, and it must not exceed the value for `totalDigits`, if one exists.

The `totalDigits` facet can be applied to `decimal` or any of the integer types, as well as types derived from them. The `fractionDigits` facet may only be applied to `decimal`, because it is fixed at 0 for all integer types.

### 8.4.4 Enumeration

The enumeration facet allows you to specify a distinct set of valid values for a type. Unlike most other facets (except `pattern` and `assertion`), the enumeration facet can appear multiple times in a single restriction. Each enumerated value must be unique, and must be valid for that type. If it is a string-based or binary type, you may also specify the empty string in an enumeration value, which allows elements or attributes of that type to have empty values.

Example 8–9 shows a simple type `SMLXSizeType` that allows the values `small`, `medium`, `large`, and `extra large`.

#### **Example 8–9.** Applying the enumeration facet

---

```
<xs:simpleType name="SMLXSizeType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
    <xs:enumeration value="extra large"/>
  </xs:restriction>
</xs:simpleType>
```

---

When restricting types that have enumerations, it is important to note that you must *restrict*, rather than *extend*, the set of enumeration values. For example, if you want to restrict the valid values of `SMLSizeType` to only be `small`, `medium`, and `large`, you could define a simple type as in Example 8–10.

---

**Example 8–10. Restricting an enumeration**

---

```
<xs:simpleType name="SMLSizeType">
  <xs:restriction base="SMLXSizeType">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>
```

---

Note that you need to repeat all of the enumeration values that apply to the new type. This example is legal because the values for `SMLSizeType` (`small`, `medium`, and `large`) are a subset of the values for `SMLXSizeType`. By contrast, Example 8–11 attempts to add an enumeration facet to allow the value `extra small`. This type definition is illegal because it attempts to extend rather than restrict the value space of `SMLXSizeType`.

---

**Example 8–11. Illegal attempt to extend an enumeration**

---

```
<xs:simpleType name="XSMLXSizeType">
  <xs:restriction base="SMLXSizeType">
    <xs:enumeration value="extra small"/>
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
    <xs:enumeration value="extra large"/>
  </xs:restriction>
</xs:simpleType>
```

---

The only way to add an enumeration value to a type is by defining a union type. Example 8–12 shows a union type that adds the value

**Example 8–12.** Using a union to extend an enumeration

---

```
<xs:simpleType name="XSMLXSizeType">
  <xs:union memberTypes="SMLXSizeType">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="extra small"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

---

extra small to the set of valid values. Union types are described in detail in Section 10.2 on p. 183.

When enumerating numbers, it is important to remember that the enumeration facet works on the actual value of the number, not its lexical representation as it appears in an XML instance. Example 8–13 shows a simple type `NewSmallDressSizeType` that is based on `integer`, and specifies an enumeration of 2, 4, and 6. The two instance elements shown, which contain 2 and 02, are both valid. This is because 02 is equivalent to 2 for integer-based types. However, if the base type of `NewSmallDressSizeType` had been `string`, the

**Example 8–13.** Enumerating numeric values

---

Schema:

```
<xs:simpleType name="NewSmallDressSizeType">
  <xs:restriction base="xs:integer">
    <xs:enumeration value="2"/>
    <xs:enumeration value="4"/>
    <xs:enumeration value="6"/>
  </xs:restriction>
</xs:simpleType>
```

Valid instances:

```
<size>2</size>
<size>02</size>
```

---

value 02 would not be valid, because the strings 2 and 02 are not the same. If you wish to constrain the lexical representation of a numeric type, you should apply the `pattern` facet instead. For more information on type equality in XML Schema, see Section 11.7 on p. 253.

The `enumeration` facet can be applied to any type except `boolean`.

### 8.4.5 *Pattern*

The `pattern` facet allows you to restrict values to a particular pattern, represented by a regular expression. Chapter 9 provides more detail on the rules for the regular expression syntax. Unlike most other facets (except `enumeration` and `assertion`), the `pattern` facet can be specified multiple times in a single restriction. If multiple `pattern` facets are specified in the same restriction, the instance value must match at least one of the patterns. It is not required to match all of the patterns.

Example 8–14 shows a simple type `DressSizeType` that includes the pattern `\d{1,2}`, which restricts the size to one or two digits.

#### **Example 8–14.** Applying the pattern facet

---

```
<xs:simpleType name="DressSizeType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="2"/>
    <xs:maxInclusive value="18"/>
    <xs:pattern value="\d{1,2}"/>
  </xs:restriction>
</xs:simpleType>
```

---

When restricting types that have patterns, it is important to note that you must *restrict*, rather than *extend*, the set of valid values that the patterns represent. In Example 8–15, we define a simple type `SmallDressSizeType` that is derived from `DressSizeType`, and add an additional `pattern` facet that restricts the size to one digit.

**Example 8–15.** Restricting a pattern

---

```
<xs:simpleType name="SmallDressSizeType">
  <xs:restriction base="DressSizeType">
    <xs:minInclusive value="2"/>
    <xs:maxInclusive value="6"/>
    <xs:pattern value="\d{1}"/>
  </xs:restriction>
</xs:simpleType>
```

---

It is not technically an error to apply a pattern facet that does not represent a subset of the ancestors' pattern facets. However, the schema processor tries to match the instance value against the pattern facets of both the type and its ancestors, ensuring that it is in fact a subset. Example 8–16 shows an illegal attempt to define a new size type that allows the size value to be up to three digits long. While the schema is not in error, it will not have the desired effect because the schema processor will check values against both the pattern of `LongerDressSizeType` and the pattern of `DressSizeType`. The value `004` would not be considered a valid instance of `LongerDressSizeType` because it does not conform to the pattern of `DressSizeType`.

**Example 8–16.** Illegal attempt to extend a pattern

---

```
<xs:simpleType name="LongerDressSizeType">
  <xs:restriction base="DressSizeType">
    <xs:pattern value="\d{1,3}"/>
  </xs:restriction>
</xs:simpleType>
```

---

Unlike the enumeration facet, the pattern facet applies to the lexical representation of the value. If the value `02` appears in an instance, the pattern is applied to the digits `02`, not `2` or `+2` or any other form of the integer.

The pattern facet can be applied to any type.

### 8.4.6 *Assertion*

The `assertion` facet allows you to specify additional constraints on values using XPath 2.0. Example 8–17 is a simple type with an assertion, namely that the value must be divisible by 2. It uses a facet named `assertion` with a `test` attribute that contains the XPath expression.

Simple type assertions are a flexible and powerful feature covered in more detail, along with complex type assertions, in Chapter 14.

---

**Example 8–17.** Simple type assertion

---

```
<xs:simpleType name="EvenDressSizeType">
  <xs:restriction base="DressSizeType">
    <xs:assertion test="$value mod 2 = 0" />
  </xs:restriction>
</xs:simpleType>
```

---

### 8.4.7 *Explicit Time Zone*

The `explicitTimezone` facet allows you to control the presence of an explicit time zone on a date/time value. Example 8–18 is a simple type based on `time` but with an explicit time zone required. The syntax of time zones is described in more detail in Section 11.4.13 on p. 233.

The `value` attribute of `explicitTimezone` has three possible values:

1. `optional`, making the time zone optional (the value for most built-in date/time types)
2. `required`, making the time zone required (the value for the `dateTimeStamp` built-in type)
3. `prohibited`, disallowing the time zone

---

**Example 8–18.** Explicit time zone

---

```
<xs:simpleType name="SpecificTimeType">
  <xs:restriction base="xs:time">
    <xs:explicitTimezone value="required"/>
  </xs:restriction>
</xs:simpleType>
```

---

### 8.4.8 *Whitespace*

The `whiteSpace` facet allows you to specify the whitespace normalization rules which apply to this value. Unlike the other facets, which restrict the value space of the type, the `whiteSpace` facet is an instruction to the schema processor on to what to do with whitespace. This type of facet is known as a *prelexical* facet because it results in some processing of the value before the other constraining facets are applied. The valid values for the `whiteSpace` facet are:

- `preserve`: All whitespace is preserved; the value is not changed.
- `replace`: Each occurrence of tab (`#x9`), line feed (`#xA`), and carriage return (`#xD`) is replaced with a single space (`#x20`).
- `collapse`: As with `replace`, each occurrence of tab (`#x9`), line feed (`#xA`), and carriage return (`#xD`) is replaced with a single space (`#x20`). After the replacement, all consecutive spaces are collapsed into a single space. In addition, leading and trailing spaces are deleted.

Table 8–6 shows examples of how values of a string-based type will be handled depending on its `whiteSpace` facet.

**Table 8–6** Handling of string values depending on `whiteSpace` facet

<i>Original string</i>	string ( <i>preserve</i> )	normalizedString ( <i>replace</i> )	token ( <i>collapse</i> )
a string	a string	a string	a string
on two lines	on two lines	on two lines	on two lines
has spaces	has spaces	has spaces	has spaces
leading tab	leading tab	leading tab	leading tab
leading spaces	leading spaces	leading spaces	leading spaces

The whitespace processing, if any, will happen first, before any validation takes place. In Example 8–9, the base type of `SMLXSizeType`

is `token`, which has a `whiteSpace` facet of `collapse`. Example 8–19 shows valid instances of `SMLXSizeType`. They are valid because the leading and trailing spaces are removed, and the line feed is turned into a space. If the base type of `SMLXSizeType` had been `string`, the whitespace would have been left as is, and these values would have been invalid.

**Example 8–19.** Valid instances of `SMLXSizeType`

---

```
<size> small </size>
```

```
<size>extra  
large</size>
```

---

Although you should understand what the `whiteSpace` facet represents, it is unlikely that you will ever apply it directly in your schemas. The `whiteSpace` facet is fixed at `collapse` for most built-in types. Only the string-based types can be restricted by a `whiteSpace` facet, but this is not recommended. Instead, select a base type that already has the `whiteSpace` facet you want. The types `string`, `normalizedString`, and `token` have the `whiteSpace` values `preserve`, `replace`, and `collapse`, respectively. For example, if you wish to define a string-based type that will have its whitespace collapsed, base your type on `token`, instead of basing it on `string` and applying a `whiteSpace` facet. Section 11.2.1 on p. 205 provides a discussion of these three types.

## 8.5 | Preventing simple type derivation

XML Schema allows you to prevent derivation of other types from your type. By specifying the `final` attribute with a value of `#all` in your simple type definition, you prevent derivation of any kind

(restriction, extension, list, or union). If you want more granular control, the value of `final` can be a whitespace-separated list of any of the keywords `restriction`, `extension`, `list`, or `union`. The `extension` value refers to the extension of simple types to derive complex types, described in Section 13.4.1 on p. 306. Example 8–20 shows some valid values for `final`.

**Example 8–20.** Valid values for the `final` attribute in simple type definitions

---

```
final="#all"  
final="restriction list union"  
final="list restriction extension"  
final="union"  
final=""
```

---

Example 8–21 shows a simple type that cannot be restricted by any other type or used as the item type of a list. With this definition of `DressSizeType`, it would have been illegal to define `MediumDressSizeType` in Example 8–4 because it attempts to restrict `DressSizeType`.

**Example 8–21.** Preventing type derivation

---

```
<xs:simpleType name="DressSizeType" final="restriction list">  
  <xs:restriction base="xs:integer">  
    <xs:minInclusive value="2"/>  
    <xs:maxInclusive value="18"/>  
  </xs:restriction>  
</xs:simpleType>
```

---

If no `final` attribute is specified, it defaults to the value of the `finalDefault` attribute of the schema element. If neither `final` nor `finalDefault` is specified, there are no restrictions on derivation from that type. You can specify the empty string ("") for the `final` value if you want to override the `finalDefault` value.

## 8.6 | Implementation-defined types and facets

Starting with version 1.1, additional simple types and facets may be defined and supported by a particular XML Schema implementation.

### 8.6.1 *Implementation-defined types*

An implementation can choose to support a set of primitive simple types in addition to those built into XML Schema (described in Chapter 11).

Suppose that an implementation defines a special primitive type `ordinalDate` that represents an ordinal date: a year, followed by a hyphen, followed by a number from 001 to 366 indicating the day of the year. Although an ordinal date value could be represented as a string, it may be beneficial to promote it to its own primitive type if it has special considerations for ordering or validation of its values, or special operations that can be performed on it (for example, subtracting two ordinal dates to get a duration).

A schema author can use an implementation-defined type just like a built-in type, except that it will be in a different namespace defined by the implementation. The schema in Example 8–22

#### **Example 8–22.** Using an implementation-defined type

---

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:ext="http://example.org/extensions">
  <xs:element name="anyOrdinalDate" type="ext:ordinalDate"/>
  <xs:element name="recentOrdinalDate" type="OrdinalDateIn2011"/>
  <xs:simpleType name="OrdinalDateIn2011">
    <xs:restriction base="ext:ordinalDate">
      <xs:minInclusive value="2011-001"/>
      <xs:maxInclusive value="2011-365"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

---

contains two references to the `ordinalDate` type, which is in the hypothetical `http://example.org/extensions` namespace. The `anyOrdinalDate` element declaration refers to the type directly by its qualified name. The `OrdinalDateIn2011` user-defined simple type is a restriction of `ordinalDate` using bounds facets to specify a range of allowed values.

### 8.6.2 *Implementation-defined facets*

Implementation-defined facets might specify additional constraints on the valid values, or even signal to the processor how to process the value. An example is the Saxon processor's `preprocess` facet which allows you to specify an XPath expression that transforms the value in some way before validation.

In Example 8–23, the `saxon:preprocess` facet appears among the children of `restriction`. You can tell that it is an implementation-defined facet because it is in a different namespace, `http://saxon.sf.net/`. This particular example is telling the processor to convert the value to upper case before validating it against the enumeration facets. It is essentially implementing a case-insensitive enumeration.

#### **Example 8–23.** Using the Saxon preprocess facet

---

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:saxon="http://saxon.sf.net/">
  <xs:simpleType name="SMLXSizeType">
    <xs:restriction base="xs:token">
      <saxon:preprocess action="upper-case($value)"/>
      <xs:enumeration value="SMALL"/>
      <xs:enumeration value="MEDIUM"/>
      <xs:enumeration value="LARGE"/>
      <xs:enumeration value="EXTRA LARGE"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

---

Implementation-defined facets can apply to the XML Schema built-in types (and user-defined restrictions of them); they can also apply to any implementation-defined types such as the `ordinalDate` example type described in the previous section.

While implementation-defined types and facets can be useful, they do affect the portability of your schema. With the schema in Example 8–23, if you try to validate a document that contains lower-case “small” for a size, it would be valid when using Saxon but not when using a different implementation. Therefore, implementation-defined facets should be used only in controlled situations. Section 23.5.3 on p. 642 provides more information on how to make your schemas more portable across implementations when using implementation-defined types and facets.



# Index

## Symbols

- `_` (underscore)
  - in NMTOKEN type, 243
  - in XML names, 40, 91, 167, 208, 559
- `-` (hyphen, dash, minus sign)
  - in binary types, 249
  - in dates, 221–228, 234
  - in durations, 229–233
  - in NMTOKEN type, 243
  - in numbers, 214–219
  - in regular expressions, 161, 165, 172–176
  - in time values, 233–234
  - in XML names, 40, 91, 167, 208, 559
  - in XPath 2.0, 356
- `,` (comma)
  - in DTDs, 477
  - in regular expressions, 176–177
- `;` (semicolon), in regular expressions, 162
- `:` (colon)
  - in NMTOKEN type, 243
  - in time values, 222–225, 233–234
  - in XML names, 167, 208–209, 246, 559
- `!=` operator (XPath 2.0), 356, 378
- `?` (question mark)
  - in DTDs, 477
  - in regular expressions, 162, 165, 176–177
- `/` (slash)
  - in binary types, 249
  - in XPath, 436–437
- `//` (XPath), 438
- `.` (period)
  - in NMTOKEN type, 243
  - in numbers, 215–216
  - in regular expressions, 162, 165, 175
  - in time values, 222–225
  - in XML names, 40, 91, 167, 209, 559
  - in XPath, 436–437
- `./` (XPath), 436–437
- `^` (circumflex accent), in regular expressions, 165, 173–175
- `'` (apostrophe), in regular expressions, 163
- `"` (quote), in regular expressions, 163
- `( )` (parentheses)
  - in DTDs, 476–477
  - in regular expressions, 161–162, 165, 175–178
  - in XPath 2.0, 355
- `[ ]` (square brackets), in regular expressions, 160–162, 165, 171–177

---

Index entries in gray refer to XML Schema 1.1.

- { } (curly brackets), in regular expressions, 160–162, 165, 176–177
  - @ (commercial at), in XPath, 436–437
  - \* (asterisk)
    - in DTDs, 474, 477
    - in regular expressions, 162–163, 165, 176–177
    - operator (XPath 2.0), 356**
    - wildcard (XPath), 436–437
  - \ (backslash), in regular expressions, 161–166, 175
  - & (ampersand)
    - in character entities, 206
    - in regular expressions, 163
    - in strings, 205
  - # (number sign)
    - in regular expressions, 162
    - in URIs, 251–252
  - % (percent sign), in URIs, 251–252
  - + (plus sign)
    - in binary types, 249
    - in DTDs, 477
    - in numbers, 104, 214–219, 254
    - in regular expressions, 162, 165–166, 176–177
    - in time values, 234
    - in XPath 2.0, 356**
  - < (less than), 163, 206
    - in regular expressions, 163
    - in strings, 205
    - in XPath 2.0, 356, 378**
  - <= **operator (XPath 2.0), 356, 378**
  - = (equals sign)
    - in binary types, 249
    - in XPath 2.0, 356, 378**
  - > (greater than), 163
    - in regular expressions, 163
    - in XPath 2.0, 356, 378**
  - >= **operator (XPath 2.0), 356, 378**
  - | (vertical bar)
    - in DTDs, 474, 477–478
    - in regular expressions, 160–162, 165, 176–178
    - in XPath, 435, 437
  - 0 (negative zero), numeric value, 213–214
  - 0 (zero)
    - as boolean value, 247
    - as numeric value, 213–214
    - in dates, 221, 226
    - leading/trailing, in numbers, 104, 125, 215–217, 219, 254
  - 1 (boolean value), 247
- ## A
- a element (HTML), 525
  - abstract attribute
    - of complexType element, 343, 346–348
    - of element element, 418, 420
    - syntax of, 671
  - all group, 26, 276–278
    - avoiding in web services, 548
    - element declarations/references in, 276, 391
    - extending, 310–311, 606**
    - group references in, 276, 391**
    - in complex types, 390
    - in named model groups, 386, 391
    - occurrence constraints in, 276–277, 532
    - restricting, 325–328
      - in version 1.1, 625**
    - syntax of, 277, 649
    - vs. DTDs, 477
    - wildcards in, 276**
  - #all value
    - of block attribute, 344, 420
    - of final attribute, 152, 343
    - in version 1.1, 418**
  - alternative element, 97
    - syntax of, 376, 650
    - test attribute of, 375–376
    - type attribute of, 376
    - xpathDefaultNamespace attribute of, 375, 381

- &amp; entity reference
  - in regular expressions, 163
  - in strings, 205
- ancestor keyword (XPath), 438
- and operator (XPath 2.0)**, 356, 378
- annotation element, 31, 497–498, 581, 585
  - syntax of, 581, 650
  - vs. DTDs, 497–498
  - vs. non-native attributes, 589
- anonymous complex types, 22, 96–97, 260–261
  - never equal, 269
  - vs. named, 550
- anonymous simple types, 22, 96
  - definitions of, 120–121, 132–133
  - readability of, 134
  - vs. named, 133–134, 550
- ANY specifier (DTDs), 474, 480
- any wildcard, 27, 285–288, 480, 601
  - namespace attribute of, 287, 329
  - notNamespace attribute of, 289–290
  - notQName attribute of, 290
  - processContents attribute of, 287–289
  - syntax of, 285, 650
- ##any value (namespace attribute on wildcard), 287, 329
- anyAtomicType type, 203
- anyAttribute wildcard, 27, 298, 398, 602
  - namespace attribute of, 298, 315–316, 336–337
  - notNamespace attribute of, 298
  - notQName attribute of, 298
  - processContents attribute of, 298, 315–316, 336–337, 602–603
  - syntax of, 299, 651
- anySimpleType type, 121, 203
- anyType type, 96, 203
- anyURI type, 250–253
  - facets applicable to, 253, 698
  - values of, 143–144
- &apos; entity reference, 163
- appinfo element, 31, 581, 587–588
  - syntax of, 587, 651
- applications
  - compatibility of, 634–635
  - ignoring irrelevant elements and attributes in, 635
  - providing information for, 6, 31
- appliesToEmpty attribute (defaultOpenContent element), 298
  - syntax of, 672
- assert element, 365–372
  - syntax of, 366, 651
  - test attribute of, 352, 375
  - xpathDefaultNamespace attribute of, 375
- assertion element, 137, 150
  - for built-in types, 695–698
  - syntax of, 353, 652
  - test attribute of, 352, 375
  - xpathDefaultNamespace attribute of, 375
- assertions, 33, 351–375
  - and namespaces, 372–375
  - for complex types, 365–372
  - for list types, 363–365
  - for simple types, 353–365
  - for union types, 185
  - inheriting, 362, 370–372
  - multiple on the same type, 355
  - type errors in, 359–362
  - using XPath 2.0 for, 352
  - with multistep paths, 368
  - with predicates, 368
- assisted editing, 541
- asterisk. *See* \*
- atomic types, 181–183
- atoms, in regular expressions, 161–176
- attribute declarations, 4, 18, 115–122, 281–284
  - from a different namespace, 117
  - global, 115–117, 119–120, 394, 550
  - local, 19, 117–120, 122, 281, 339, 394, 550

- attribute declarations (*cont.*)
  - location of, in type definitions, 281
  - overriding**, 459, 466
  - removing, 619
  - restricting, 318, 333–335
  - reusing, 597
  - target namespace of, 48
  - vs. wildcards, 619
- attribute element, 115–125
  - default attribute of, 123
  - fixed attribute of, 123
  - form attribute of, 122
  - inheritable attribute of**, 126–127
  - name attribute of, 117
  - ref attribute of, 115, 117
  - syntax of:
    - global declaration, 116, 652
    - local declaration, 118, 652
    - reference, 282, 653
  - targetNamespace attribute of**, 339–341
  - type attribute of, 120–121, 394
  - use attribute of, 117, 119, 283, 394, 482–483, 688
- attribute group references, 395–399
  - in attribute groups, 398–399
  - in complex type definitions, 284, 396
  - location of, in type definitions, 281
- attribute groups, 19, 32, 120, 392–400, 544
  - attribute references in, 394
  - default**, 284, 399–400
  - definitions of, 18, 393
  - local attribute declarations in, 394
  - location of, in type definitions, 281
  - names of, 393, 545, 559
    - duplicating (illegal), 397
  - order of, 396
  - overriding**, 459, 467, 491–492, 613–614
  - risks of**, 470–471
  - redefining, 449, 451, 456–458, 490–491, 600, 611–612
  - risks of, 470–471
  - reusing, 597
  - vs. complex type extension, 403
  - vs. DTDs, 485–486
  - wildcards in, 395, 398
- attribute references, 282–283
  - in attribute groups, 394
  - location of, in type definitions, 281
- attribute wildcards, 27, 298, 602
  - extending complex types with, 315
  - for forward compatibility, 623
  - in attribute groups, 395, 398
  - processContents attribute of, 315, 336
  - restricting, 335–337, 458
  - vs. attribute declarations, 619
  - See also* anyAttribute element
- attributeFormDefault attribute (schema element), 77, 122
  - ignoring for attribute names, 580
  - qualified value of, 122
  - syntax of, 672
  - unqualified value of, 122
- attributeGroup element, 393
  - ref attribute of, 395
  - syntax of:
    - definition, 393, 653
    - reference, 396, 653
- attributes, 19
  - and default namespace declarations, 39
  - applicable to all elements, 79–81
  - co-constraints for**, 33
  - deprecated, 627, 638
  - empty, 124, 145
  - extending, 314, 490–491
  - inherited**, 126–127, 283, 382–383
  - names of, 4, 117, 545, 559–563
    - changing, 619
    - duplicating (illegal), 45, 119, 397, 470
    - in XPath**, 367, 436–437, 439
    - qualified (prefixed), 44–46, 117, 119–120, 122–123, 580
    - unprefixed, 44, 122
    - unqualified, 40, 119, 122–123, 580

- nillable, 435
  - non-native, 588–589, 629
  - order of, 115, 281
  - prohibited, 333, 335
  - required vs. optional, 117, 119, 123, 143, 283, 333, 457, 482, 618–619, 627
  - reusing, 302
  - types associated with, 4, 18, 21, 120–121, 125, 236–246, 457
    - enumerated, 481
    - vs. DTDs, 480–482
  - units of measurement in, 130
  - values of:
    - default, 82, 114, 123–124, 283, 333, 457, 482, 499, 548
    - fixed, 82, 123, 125, 283, 333, 457, 548
    - validating, 114
    - whitespace in, 82
  - vs. elements, 113
  - avg function (XPath 2.0), 363**
- B**
- b element (HTML), 525
  - backslash, in regular expressions, 161–166, 175
  - backward compatibility, 618–622
    - and version numbers, 626, 631–632
  - base attribute (restriction) element, 136
    - syntax of, 672
  - base64 encoding, 249
  - base64Binary type, 248–250
    - facets applicable to, 250, 698
    - values of:
      - empty, 143
      - length of, 249
  - BIDI elements, 207
  - binary types, 248–250, 695
    - facets applicable to, 250, 698
    - values of:
      - length of, 143–144, 249
      - valid, 145
  - block attribute
    - #all value of, 344, 420
    - empty value of, 345
    - extension value of, 344, 420
    - of complexType element, 343–346
    - of element element, 322, 346, 418–419
    - restriction value of, 344, 420
    - substitution value of, 420
    - syntax of, 672–673
  - block element (XSL-FO), 525
  - block escapes, 170–171
  - blockDefault attribute (schema element), 77, 345
    - syntax of, 673
  - boolean type, 247–248, 255
    - facets applicable to, 698
  - bounds facets, 142
    - applicable to:
      - date and time types, 235
      - duration types, 230
      - integer types, 217–218
      - simple content, 317–318
    - changing restrictiveness of, 619–620
  - br element (HTML), 265
  - branches (in regular expressions), 159–161, 177–178
  - built-in types, 23, 201–255, 691–695
    - facets applicable to, 138–139, 141, 152, 695–698
    - hierarchy of, 202–203
    - namespace of, 50
    - restricting, 135, 140
  - byte type, 218
    - facets applicable to, 696
- C**
- C, in category escapes, 169
  - \c and \C, multicharacter escapes, 167
  - camel case, 560
  - carriage return character
    - in regular expressions, 163, 165–166
    - in strings, 151
  - CDATA data type specifier (DTDs), 481

- chameleon namespaces, 65–66, 565, 572–574
- character class escapes, 161, 164
  - block, 170–171
  - category, 167–169
  - multicharacter, 166
  - single-character, 165
- character class expressions, 161, 171
  - escaping, 175
  - list of characters in, 171
  - negating, 173–174
  - ranges in, 172
  - subtracting from, 174
- character references, 162–163
- chatty services, 522
- choice group, 26, 273–276
  - avoiding in web services, 548
  - compatibility of, 619
  - element wildcards in, 609
  - extending, 309, 489–490, 607–608
  - in complex types, 390
  - in named model groups, 386
  - meaningless, 321
  - nesting, 275–276
  - occurrence constraints in, 274–275
  - restricting, 324–329
  - syntax of, 273, 654
  - vs. DTDs, 477
  - vs. substitution groups, 414–415
- circumflex accent, in regular expressions, 165, 173–175
- class hierarchies, 518
- co-constraints, 586
- code generating, 541
- collapse value (`whiteSpace` facet), 104, 125, 151–152, 189, 205–206, 255
- colon. *See* :
- comma. *See* ,
- comments, 497–498
- common components libraries, 597–598
- compatibility, 617–626
  - application, 634–635
  - backward, 618–622, 626, 631–632
  - forward, 623–626
- complex content
  - deriving complex types from, 304–305
  - extending, 307–316
  - restricting, 318–329
    - in version 1.1**, 320
  - vs. database tables, 504
  - vs. DTDs, 476
- complex type definitions
  - attribute group references in, 396
  - attribute groups in, 394, 397
  - attribute wildcards in, 298, 398, 602
  - attributes in, 115–119, 281, 283–284, 397
  - element declarations in, 93, 266–268, 387
  - element wildcards in, 601
  - ID attributes in, 236
  - named model groups in, 278–279, 387, 389
  - repeating element names in, 415–416
- complex types, 19, 257–298
  - abstract, 346, 348
  - anonymous, 22, 96–97, 260–261, 269, 550
  - assertions for**, 365–372
  - associating with element names, 96
  - base type of, 302, 314–315, 319
  - extending, 22, 27, 302–303, 305–316, 403, 606
  - global vs. local, 19, 22
  - named, 258–260, 484–485, 550, 559–560, 597
  - overriding**, 459, 465
  - risks of**, 468–470
  - preventing derivation of, 343–344
  - redefining, 449, 451, 453–454, 600, 609
    - risks of, 468–470
  - restricting, 22, 27, 301–303, 316–337, 455, 603
  - reusing, 597
  - vs. named model groups, 403–404
  - vs. OO classes, 514

- vs. simple types, 21
- with open content**, 604
- `complexContent` element, 304
  - syntax of, 304, 654
- `complexType` element, 28, 96, 258–261
  - abstract attribute of, 343, 346–348
  - block attribute of, 343–346
  - `defaultAttributes` attribute of, 400
  - final attribute of, 343–344
  - mixed attribute of, 265
  - no direct element declarations in, 270
  - syntax of:
    - anonymous definition, 261, 655
    - named definition, 259, 654
- composition, 519–522
- conditional expressions (XPath)**, 369–370
- conditional inclusion, 642
- conditional type assignment**, 375–383
  - and namespaces, 381–382
  - inherited attributes in, 382–383
  - validation errors in, 380–381
  - with declared types, 378
  - with default types, 377
- `contains` function (XPath 2.0), 357
- content models, 26, 261–269
  - absence of, for empty content, 479
  - and web services, 548
  - deterministic, 279–280
  - eliminating meaningless groups in, 320–321
  - extending, 27, 305–313, 607
  - in DTDs, 473–480
  - location of extensions in, 600, 611
  - named model groups in, 390
  - nondeterministic, 280, 470, 602
  - open, 27, 292–298, 311, 600, 619
  - restricting, 318–333, 455
  - reusability of, 302, 385
  - reusing, 484–485
- content types, 25–26, 266, 473–480
  - See also* complex, element-only, empty, mixed, simple content

- Coordinated Universal Time. *See* UTC
- Costello, Roger, 550
- `count` function (XPath 2.0), 363
- curly brackets, in regular expressions, 160–162, 165, 176–177

## D

- D, in durations, 229–233
- `\d` multicharacter escape, 160–161, 166–167, 173
- `\D` multicharacter escape, 166
- dash. *See* -
- data binding tools
  - and generic elements, 520
  - complex types in, 517–518
  - processing relationships in, 511
- databases
  - coupling with XML messages, 504
  - foreign keys in, 32, 430, 510
  - generating instances from, 82
  - mapping elements to, 586–587
  - names in, 560
  - tables and columns in, 504
- datatypes, 201
- date and time types, 221–235, 693
  - comparing dates in, 235
  - facets applicable to, 142, 235, 696–697
- date type, 221
  - facets applicable to, 696
- dateTime type, 223–224
  - facets applicable to, 697
- dateTimeStamp type, 224–225
  - facets applicable to, 697
- dayTimeDuration type, 232–233
  - facets applicable to, 697
- DCD (Document Content Description), 11
- DDML (Document Definition Markup Language), 11
- debugging, 6, 542
- decimal point, 215–216
- decimal type, 145, 213–216
  - canonical representation of, 215
  - facets applicable to, 216, 696

- declarations, 18
  - See also* attribute, element, namespace declarations
- default attribute
  - of attribute element, 123, 283, 482
  - of element element, 101
  - syntax of, 673
- default values
  - avoiding in web services, 548
  - for nils, 107
  - of attributes, 82, 114, 123–124, 283, 333, 457, 482, 499
  - of elements, 82, 101–103, 110, 269
  - of occurrence constraints, 477
- defaultAttributes attribute** (schema element), 284, 399–400
  - syntax of, 674
- defaultAttributesApply attribute** (schema element), 284, 400
  - syntax of, 674
- ##defaultNamespace value** (xpathDefaultNamespace attribute), 375
- defaultOpenContent element**, 295–298
  - mode attribute of, 296
  - syntax of, 295, 655
- ##defined value (notQName attribute)**, 290–291, 625
- ##definedSibling value (notQName attribute)**, 290–291, 625
- definitions, 18
  - order of, 19
  - See also* complex type definitions
- deprecated element, 638
- derivation. *See* type derivation
- descendant keyword (XPath), 438
- deterministic content models, 279–280
- digits
  - in binary types, 249
  - in NMTOKEN type, 243
  - in regular expressions, 161–162, 166, 168
  - in XML names, 40, 91, 208, 559
- distinct-values function** (XPath 2.0), 363
- div operator** (XPath 2.0), 356
- DocBook, 526
- documentation, 6–7, 31, 580–592
  - generating from schemas, 541, 545, 584
  - human-readable, 497–498, 541, 545, 581
  - metadata in, 582, 585
  - on namespaces, 589–592
  - reusing, 584
  - separate authoring, 527
- documentation element, 497–498, 581–584
  - source attribute of, 582
  - syntax of, 583, 655
- documents. *See* schema documents
- double type, 213–215
  - canonical representation of, 213
  - facets applicable to, 215, 696
- DTDs (Document Type Definitions), 9–10, 473–499
  - #PCDATA in, 474–475, 478
  - attributes in, 475
    - types of, 480–482
    - values of, 114, 482
  - comments in, 497–498
  - converting to schemas, 10
  - elements in, 476, 478
  - empty content in, 479
  - extensibility of, 486–492
  - general entities in, 493, 499
  - groups in, 476–484
  - limited support of namespaces in, 564
  - occurrence constraints in, 477
  - parameter entities in, 483–492
  - reusing:
    - attributes, 485–486
    - content models, 484–485
  - unparsed entities in, 240, 242, 493, 496
  - using with schemas, 499
  - whitespace handling in, 474

- wildcards in, 480
  - duration type, 229–231
    - facets applicable to, 697
- E**
- e or E (exponent), in numbers, 213–214
  - e-book readers, 524
  - element declarations, 4, 18, 89–110
    - abstract, 420
    - duplicating names of, 268–269
    - global, 19, 89–92, 550, 578
    - identity constraints in, 425
    - in content models, 266–269
    - in model groups, 270, 276–279, 387, 391
    - in substitution groups, 95, 114
      - multiple, 413–414**
    - local, 19, 93–96, 99, 266, 339, 550, 578
    - missing external components in, 76
    - occurrence constraints in, 94
    - overriding, 459, 466**
    - referencing types in, 96
    - removing, 619
    - restricting, 321–322
    - reusing, 597
    - target namespace of, 48
    - vs. OO instance variables, 514
    - vs. wildcards, 280, 604, 619, 624
  - element element, 28, 89–95
    - abstract attribute of, 418, 420
    - block attribute of, 322, 346, 418–419
    - default attribute of, 101
    - final attribute of, 418–419
    - fixed attribute of, 101
    - form attribute of, 100
    - name attribute of, 266, 339
    - nillable attribute of, 109
    - ref attribute of, 267, 388
    - substitutionGroup attribute of, 410–412
  - syntax of:
    - global declaration, 90–91, 656
    - local declaration, 93, 656
    - reference, 267, 656
  - targetNamespace attribute of, 339–341**
  - type attribute of, 96–97, 267, 387
  - element references, 267–268
    - duplicating names of, 268–269
    - in model groups, 270, 276–279, 388, 391
    - occurrence constraints in, 92
  - element substitution groups. *See* substitution groups
  - element wildcards, 27, 285–288, 601
    - for forward compatibility, 623
    - in choice groups, 609
    - overlapping values of, 280
    - restricting, 322–324
    - vs. element declarations, 604, 619**
      - in version 1.1, 624**
      - in version 1.1., 280
    - See also* any, replacement elements
  - elementFormDefault attribute (schema element), 77, 99, 101, 578
    - overriding, 100**
    - qualified value of, 99, 402, 576
    - syntax of, 674
    - unqualified value of, 77, 99, 101, 577–578
  - element-only content, 25, 264
    - extending, 305
    - restricting, 317
  - elements, 19, 89–110
    - absent, 101, 105
    - block vs. inline, 525
    - co-constraints for, 33**
    - container, 531–532
    - deprecated, 627, 638
    - empty, 101, 105–106, 143, 145
    - for relationships, 512–514
    - generic, 520

- elements (*cont.*)
- intermediate, 527–531, 546
  - mapping to databases, 586–587
  - names of, 4, 96, 545, 559–563, 597
    - changing, 619
    - generic vs. specific, 533–537
    - in XPath, 439
    - qualified, 91, 98–100, 575, 580
    - searching in content, 525
    - unprefixed, 39, 94
    - unprefixed qualified, 40
    - unqualified, 40, 96, 576–580
    - unqualified local, 98–101
    - vs. names of rows in databases, 506
  - nillable, 103, 106–110, 115, 143, 435
    - and identity constraints, 110
    - of derived declarations, 322
  - order of, 115, 270–272, 506, 536, 619
    - in narrative content, 526
  - repeating, 114, 506, 536
    - in all groups, 532
  - replacement, 285–291, 323
  - required vs. optional, 143, 271, 531, 536, 618–619, 627
  - root, 87, 95, 523
  - separate vs. composite values, 532–533
  - types associated with, 4, 18, 20–21, 96
  - values of:
    - and units of measurement, 130
    - data types for, 535
    - default, 82, 101–103, 107, 110, 269, 548
    - fixed, 82, 101–104, 110, 321, 548
    - whitespace in, 82
  - vs. attributes, 114
  - else keyword (XPath 2.0), 369–370**
  - empty attribute values, 124, 145
  - empty content, 25–26, 265–266
    - applying default open content to, 298**
    - extending, 305, 313
    - restricting, 317, 332–333
    - vs. DTDs, 479
  - empty function (XPath 2.0), 358**
  - EMPTY specifier (DTDs), 474, 479
  - ends-with function (XPath 2.0), 358**
  - ENTITIES type, 242–243
    - facets applicable to, 243, 697
    - in DTDs, 481
  - ENTITY type, 240–241
    - facets applicable to, 241, 697
    - in DTDs, 481, 483–492
  - enumeration facet, 137, 145–148, 481
    - case-insensitive, 155
    - extending, 146–147
    - for built-in types, 695–698
    - for derived types, 140
    - for list types, 192–194, 240, 243–244
    - for union types, 185
    - syntax of, 138, 657
  - error built-in type (XML Schema namespace), 380–381**
  - escapes. *See* character class escapes
  - exists function (XPath 2.0), 358**
  - explicitTimezone facet, 137–138, 150**
    - for built-in types, 695–698**
    - syntax of, 657**
    - value attribute of, 150, 234**
  - exponent, in numbers, 213–214
  - extensibility, 8, 542–543, 599–614
    - and anonymous types, 134
    - in DTDs, 486–492
    - intermediate elements for, 531
  - extension
    - of all groups, 606**
    - of attributes, 314, 490–491
    - of choice groups, 489–490, 607–608
    - of complex types, 22, 27, 302–303, 305–316, 403, 516, 606
    - of content models, 27, 305–313, 607
    - of enumerations, 146–147
    - of schemas, 8, 531, 542–543, 599–614
    - of sequence groups, 486, 488, 606–607

- extension element, 303, 305, 625
  - syntax for:
    - complex content, 307, 658
    - simple content, 306, 657
- extension value
  - of block attribute, 344, 420
  - of final attribute, 153, 343, 419
- F**
- facetAvailable attribute (Version Control namespace), 645–646**
  - syntax of, 675
- facets, 24, 135–152
  - applicability of, 138, 204, 695
  - changing restrictiveness of, 619–620
  - fixed, 140–141
  - implementation-defined, 155, 642
  - inheriting, 139, 204
  - order of, 136
  - prelexical, 151, 643, 646
- facetUnavailable attribute (Version Control namespace), 645–646**
  - syntax of, 675
- false function (XPath 2.0), 358**
- field element
  - syntax of, 435, 658
  - xpath attribute of, 435
  - xpathDefaultNamespace attribute of, 375**
- final attribute
  - #all value of, 152, 343**
    - in version 1.1, 418**
  - empty value of, 343, 419
  - extension value of, 153, 343, 419
  - list value of, 153
  - of complexType element, 343–344
  - of element element, 418–419
  - of simpleType element, 152–153
  - restriction value of, 153, 343, 419
  - syntax of:
    - on complex type, 675
    - on element, 676
    - on simple type, 676
  - union value of, 153
- finalDefault attribute (schema element), 77, 153, 344
  - overriding, 153, 419**
  - syntax of, 676
- fixed attribute
  - of attribute element, 123, 283, 482, 631
  - of element element, 101
  - of facets, 140–141
  - syntax of:
    - on declaration, 677
    - on facet, 677
- fixed values
  - avoiding in web services, 548
  - of attributes, 82, 123, 125, 283, 333, 457
  - of elements, 82, 101–104, 110, 321
  - of schema's version, 631
- #FIXED specifier (DTDs), 483**
- float type, 213–215
  - canonical representation of, 213
  - facets applicable to, 215, 696
- floating-point numbers, 213
- form attribute
  - ignoring for attribute names, 580
  - of attribute element, 122
  - of element element, 100
  - qualified value of, 100, 122
  - syntax of, 677
  - unqualified value of, 100, 122
- forward compatibility, 623–626
  - in version 1.1, 625, 641–642**
- fractionDigits facet, 137, 145**
  - applicability of, 138
  - fixed, 141
  - for built-in types, 695–698
  - for numeric types, 219
  - syntax of, 138, 658
- fragment identifiers, in URIs, 251
- G**
- Garden of Eden design, 555–557
- gDay type, 228**
  - facets applicable to, 697

- general entities (DTDs), 493, 499
- gMonth type, 227
  - facets applicable to, 697
- gMonthDay type, 227–228
  - facets applicable to, 697
- granularity of data, 130
- Gregorian calendar, 221, 225
- group element, 386–390
  - ref attribute of, 388–389, 519
  - syntax of:
    - definition, 387, 659
    - reference, 389, 659
- group references, 388
  - in all groups**, 276, 391
  - in complex types, 387, 390
  - nested, 392
  - occurrence constraints in, 386, 390
  - self-referencing, 456
- &gt; entity reference, 163
- gYear type, 225
  - facets applicable to, 697
- gYearMonth type, 226
  - facets applicable to, 697
- H**
- H, in durations, 229–233
- hexadecimal encoding, 162, 248
- hexBinary type, 248–250
  - facets applicable to, 250, 698
  - values of:
    - comparing, 255
    - empty, 143
    - length of, 249
- hyperlinks, 525
- hyphen. *See* -
- I**
- i-, in language names, 211
- \i and \I, multicharacter escapes, 167
- IANA (Internet Assigned Numbers Authority), 211
- id attribute, 678
- ID type, 236–237
  - facets applicable to, 237, 697
  - for containment relationships, 511
  - in attribute groups, 394
  - in DTDs, 481
  - limitations of, 424
  - unique values of, 236
  - using attributes for, 115
- identity constraints, 19, 32, 238, 423–444
  - and namespaces, 54, 439–440
  - changing restrictiveness of, 322
  - definitions of, 18
  - fields in, 110, 426, 434
  - for containment relationships, 511
  - location of, in element declarations, 425
  - names of, 426, 559
  - referencing**, 442–444
  - scope of, 424, 426
  - selectors in, 426, 433
  - XPath subset for, 435–438
- IDREF type, 237–238
  - comparing values of, 254
  - facets applicable to, 238, 697
  - for containment relationships, 511
  - in DTDs, 481
  - limitations of, 424
  - using attributes for, 115
- IDREFS type, 239–240
  - comparing values of, 254
  - facets applicable to, 240, 697
  - in DTDs, 481
- if keyword (XPath 2.0)**, 369–370
- #IMPLIED specifier (DTDs), 483
- import element, 30, 66–74
  - and namespaces, 49, 85, 568–572
  - chained, 70–74
  - namespace attribute of, 68
  - schemaLocation attribute of, 68, 85
  - syntax of, 67–68, 659
  - top-level location of, 68
  - vs. DTDs, 492
- include element, 29, 62–66, 450, 565–568
  - chameleon, 65–66, 565, 572–574
  - schemaLocation attribute of, 63

syntax of, 63, 660  
 top-level location of, 63  
 vs. DTDs, 492  
 indexes, for narrative content, 525  
 INF (infinity), numeric value, 213–214  
 -INF (negative infinity), numeric value,  
 213–214  
 +INF (positive infinity), numeric value,  
 213–214  
 inheritable attribute (attribute  
 element), 126–127, 283, 382–383  
 syntax of, 678  
 instances, 30, 79–87  
 augmenting, 82–83  
 upgrading with XSLT, 639  
 XHTML representation of, 587  
 xsi:schemaLocation attribute of,  
 30  
 int type, 217  
 canonical representation of, 204  
 facets applicable to, 696  
 integer type, 51, 217  
 comparing values of, 254–255  
 facets applicable to, 141, 217, 696  
 preceding sign rule for, 217  
 restricting, 142  
 whitespace in, 104, 125  
 integer types, 217–220  
 canonical representation of, 217  
 facets applicable to, 217–219  
 values of:  
 comparing, 220, 253  
 length of, 144–145  
 vs. strings, 220  
 interleave value (mode attribute), 293,  
 311  
 internationalization, 582  
 IRIs (Internationalized Resource  
 Identifiers), 251  
 Is (in block escapes), 170  
 ISO 11179 standard, 584  
 ISO 3166 standard, 211  
 ISO 639 standard, 211  
 ISO 8601 standard, 221

ISO/IEC 19757-2 standard, 12  
 item types, 182, 188  
 anonymous, 189  
 facets applicable to, 192–193  
 length of, 192  
 lists for (illegal), 196–197  
 restricting, 198  
 unions for, 196  
 whitespace in, 189, 195  
 itemType attribute, 189  
 syntax of, 678

## K

key constraints, 423, 428–429  
 changing restrictiveness of, 322  
 fields in, 426, 428–429  
 names of, 426  
 referencing, 442–444  
 scope of, 426, 429, 432  
 selectors in, 426  
 key element  
 for containment relationships, 511  
 syntax of:  
 definition, 429, 660  
 reference, 442, 660  
 key references, 423, 430–433  
 changing restrictiveness of, 322  
 fields in, 426, 431, 433  
 names of, 426  
 referencing, 442–444  
 scope of, 426, 432  
 selectors in, 426  
 keyref element  
 for containment relationships, 511  
 refer attribute of, 430  
 syntax of:  
 definition, 430, 661  
 reference, 442, 661

## L

L, in category escapes, 168  
 lang attribute (XML namespace), 59,  
 120, 211  
 syntax of, 678

- language type, 211–213
    - facets applicable to, 213, 696
  - last function (XPath 2.0)**, 364
  - lax value (processContents attribute), 288, 291, 602–603
    - in open content**, 605
  - length facet, 137, 143
    - changing restrictiveness of, 619–620
    - for binary types, 249
    - for built-in types, 695–698
    - for list types, 192, 240, 243–244
    - syntax of, 138, 661
  - letters
    - in binary types, 249
    - in NMTOKEN type, 243
    - in regular expressions, 161, 168
    - in XML names, 40, 91, 167, 208, 559
  - line feed character
    - in regular expressions, 163–166
    - in strings, 151
  - list element, 188–189
    - syntax of, 188, 662
  - list types, 181–183, 188–198
    - assertions for**, 363–365
    - comparing, 253
    - derived from string-based types, 195
    - disadvantages of, 190
    - empty, 192
    - facets applicable to, 182, 189–194, 240, 243–244, 698
    - item types of, 182, 188, 196–197
    - length of, 143, 192
    - no absent or nil items in, 189–190
    - restricting, 136, 182, 190–194
  - list value (final/finalDefault attributes), 153
  - local names, 40
  - ##local value
    - of namespace attribute, 287
    - of **notNamespace attribute**, 289
    - of **xpathDefaultNamespace attribute**, 375
  - localization, 582
  - long type, 217
    - facets applicable to, 696
  - lower-case function (XPath 2.0)**, 357
  - &lt; ; entity reference
    - in regular expressions, 163
    - in strings, 205
- M**
- M
- in category escapes, 168
  - in durations, 229–233
  - mantissa, in numbers, 213
  - marks, in regular expressions, 168
  - matches function (XPath 2.0)**, 358
  - max function (XPath 2.0)**, 363
  - maxExclusive facet, 137, 142
    - for built-in types, 695–698
    - for date and time types, 235
    - for duration types, 230
    - syntax of, 138, 662
  - maxInclusive facet, 137, 142
    - fixed value of, 140
    - for built-in types, 695–698
    - for date and time types, 235
    - for duration types, 230
    - for integer types, 217–218
    - for simple content, 317–318
    - syntax of, 138, 662
  - maxLength facet, 137, 143
    - changing restrictiveness of, 619–620
    - for built-in types, 695–698
    - for list types, 192, 240, 243–244
    - syntax of, 138, 663
  - maxOccurs attribute
    - changing restrictiveness of, 321, 324–329
    - default value of, 477
    - for replacement elements, 286
    - in all groups, 277
    - in element declarations, 94, 267, 322
    - in element references, 92, 267
    - in element wildcards, 602
    - in group references, 386, 390, 456
    - syntax of, 679

- unbounded value of:
    - in all groups, 276
    - in choice groups, 274
  - vs. DTDs, 477
  - maxVersion attribute (Version Control namespace), 641**
    - syntax of, 679
  - member types, 182, 184–187
  - memberTypes attribute, 184–185
    - syntax of, 679
  - metacharacters, 162
  - metadata, 114–115, 582, 585
    - using attributes for, 526
  - min function (XPath 2.0), 363**
  - minExclusive facet, 137, 142
    - for built-in types, 695–698
    - for date and time types, 235
    - for duration types, 230
    - syntax of, 138, 663
  - minInclusive facet, 137, 142
    - fixed value of, 140
    - for built-in types, 695–698
    - for date and time types, 235
    - for duration types, 230
    - for integer types, 217–218
    - for simple content, 317–318
    - syntax of, 138, 663
  - minLength facet, 137, 143
    - changing restrictiveness of, 619–620
    - for built-in types, 695–698
    - for list types, 192, 240, 243–244
    - syntax of, 138, 664
  - minOccurs attribute
    - changing restrictiveness of, 321, 324–329
    - default value of, 477
    - for defaulted elements, 102
    - for replacement elements, 286
    - in all groups, 276–277, 310
    - in choice groups, 275
    - in element declarations, 94, 267
    - in element references, 92, 267
    - in element wildcards, 602
    - in group references, 386, 390, 456
    - in sequence groups, 271
    - syntax of, 680
    - vs. DTDs, 477
  - minus sign. *See* -
  - minVersion attribute (Version Control namespace), 641**
    - syntax of, 680
  - missing values, 105–106
  - mixed attribute, 331, 478
    - of complexContent element, 305
    - of complexType element, 265
    - syntax of, 680
  - mixed content, 25–26, 207–208, 264
    - avoiding in web services, 548
    - default/fixed values for, 102
    - extending, 305, 312–313
    - restricting, 317, 331–332
    - vs. DTDs, 478
  - mod operator (XPath 2.0), 356**
  - mode attribute**
    - interleave value of, 293, 311
    - none value of, 293, 331
    - of defaultOpenContent element, 296
    - of openContent element, 293, 311
    - restricted, 329–330
    - suffix value of, 293, 296, 311, 329
    - syntax of, 681
  - model groups, 26, 270–279
    - definitions of, 18
    - meaningless, 320–321
    - named. *See* named model groups
    - nesting, 26, 275–276
    - restricting, 324–329
    - See also* all, choice, sequence group
  - modifiers, in regular expressions, 168
  - multicharacter escapes, 166
- N**
- N, in category escapes, 168–169
  - \n single-character escape, 163–165

- name attribute
  - of `attribute` element, 117
  - of `element` element, 92, 339, 387
  - of `group` element, 386
  - syntax of, 681
- Name type, 208–209
  - facets applicable to, 209, 695
- named complex types, 22, 258–260
  - extending, 606
  - names of, 258, 559–560
  - referencing in element declarations, 96
  - reusing, 597
  - vs. anonymous, 550
  - vs. DTDs, 484
- named model groups, 19, 32, 385–392, 544
  - all groups in, 391
  - and namespaces, 401–403
  - definitions of, 386–388
  - element declarations in, 387
  - element references in, 388
  - names of, 386, 545, 559
  - occurrence constraints in, 386
  - overriding**, 459, 467, 613–614
    - risks of**, 470–471
  - redefining, 449, 451, 454–456, 600, 611–612
    - risks of, 470–471
  - referencing, 388–392, 456
    - in complex types, 278–279, 389–391
    - in named model groups, 392
  - reusing, 597
  - target namespace of, 48
  - vs. DTDs, 476–484
  - vs. OO concepts, 519
  - vs. type derivation, 403–404, 520
  - See also* all, choice, sequence group
- named simple types, 22, 121
  - definition of, 131–132
  - in local attributes, 120
  - names of, 131, 559–560
  - referencing in element declarations, 96
  - reusing, 597
  - vs. anonymous, 133–134, 550
- names, 35–46
  - capitalization in, 560
  - case sensitiveness of, 40, 559
  - changing, 619
  - disallowed for replacement elements**, 289–291
  - duplicating (illegal), 45, 119, 397, 470
  - good practice of, 545, 559–563
  - in databases and programming languages, 560
  - length of, 560
  - non-colonized, 37, 40, 91, 210
  - prefixed, 40
  - qualified, 40, 246
    - of attributes, 44–46, 117, 119–120, 122–123, 580
    - of elements, 91
    - uniqueness of, 75–76, 117
  - qualified local, 98–100, 575, 580
  - searching in content, 525
  - separators in, 560
  - terms and abbreviations in, 561–562
  - uniqueness of, 19, 557, 568, 574
  - unprefixed, 40, 94
    - of attributes, 122
    - of elements, 39
    - undeclaring default namespace with, 43
  - unqualified, 40
    - of attributes, 119, 122–123, 580
    - of elements, 96
  - unqualified local, 98–101, 576–580
  - valid XML, 208, 559
- namespace attribute
  - `##any` value of, 287, 329
  - `##local` value of, 287
  - `##other` value of, 287, 323, 602
  - `##targetNamespace` value of, 287
  - of attribute wildcard, 298, 315–316, 336–337
  - of derived attribute declaration, 335–336

- of derived element declaration, 322
- of element wildcard, 287
- of `import` element, 68
- of restricted open content, 329–330**
- syntax of:
  - on `import`, 682
  - on wildcard, 681
- namespace declarations, 29, 37–39
  - default, 39–43
    - and attributes, 39
    - undeclaring, 43–44**
  - overriding, 42–43
  - scope of, 41
  - setting in schema documents, 52–54
- namespace-qualified names. *See* qualified names
- namespaces, 28–29, 35–54
  - advantages of, 36, 564
  - and assertions, 372–375**
  - and conditional type assignment, 381–382**
  - and imports, 66–67, 568–572
  - and includes, 565–568
  - and named model groups, 401–403
  - and schemas, 48, 565
  - chameleon, 65–66, 565, 572–574
  - default, 52, 101, 571, 578
    - for path expressions, 60**
  - disallowed for replacement elements, 289–291**
  - documenting, 589–592
  - in path expressions, 440–441
  - limited support in DTDs for, 564
  - multiple in an instance, 85
  - names of, 35–44
    - case sensitiveness of, 36
    - version numbers in, 632
  - of overridden schema documents, 459–462, 572**
  - of redefined schema documents, 448, 450, 572, 600
  - of replacement elements, 287
  - organizing, 565–574
  - prefixes for, 28, 37, 41
    - target. *See* target namespace
- Namespaces in XML recommendation, 36–37
- NaN (Not a Number), numeric value, 213–214
- narrative content, 524–527
- NCName type, 210, 236
  - comparing values of, 254
  - facets applicable to, 210, 695
- NCNames. *See* non-colonized names
- NDR (Naming and Design Rules), 548, 601
- negation, in regular expressions, 165, 173–175
- negative infinity. *See* `-INF`
- negative sign. *See* `-`
- negativeInteger type, 217
  - facets applicable to, 696
- newline. *See* line feed character
- nil attribute (XSI namespace), 51, 80, 103, 107–110, 143
  - syntax of, 682
- nil values, 103, 106–110, 115
- nillable attribute (element element), 109
  - of derived element declarations, 322
  - syntax of, 682
- NLM XML, 526
- NMTOKEN type, 243–244
  - facets applicable to, 244, 697
  - in DTDs, 481
- NMTOKENS type, 244
  - facets applicable to, 245, 697
  - in DTDs, 481
- noNamespaceSchemaLocation
  - attribute (XSI namespace), 51, 80, 83–84, 86
  - syntax of, 683
- non-colonized names (NCNames), 37, 40, 91, 210
- nondeterministic content models, 280, 470, 602
- none value (mode attribute), 293, 331
- non-native attributes, 588–589, 629

- nonNegativeInteger type, 217
    - facets applicable to, 696
  - nonPositiveInteger type, 217
    - facets applicable to, 696
  - normalizedString type, 205–208, 481
    - empty value of, 103, 143, 206
    - facets applicable to, 205–207, 695
    - whitespace in, 151–152, 205–206
  - normalize-space function (XPath 2.0), 357**
  - not function (XPath 2.0), 358, 378**
  - notation element, 494
    - syntax of, 664
  - NOTATION type, 245–246
    - facets applicable to, 246, 697
    - in DTDs, 481–482
    - restricting, 495
  - notations, 19, 245–246, 493–496
    - and unparsed entities, 496
    - declarations of, 18, 494–495
    - names of, 494, 559–563
    - overriding, 459**
    - reusing, 597
    - target namespace of, 48
  - notNamespace attribute
    - ##local value of, 289
    - ##targetNamespace value of, 289
    - of attribute wildcard, 298
    - of element wildcard, 289–290
    - syntax of, 683
  - notQName attribute
    - ##defined value of, 290–291, 625
    - ##definedSibling value of, 290–291, 625
    - of attribute wildcard, 298
    - of element wildcard, 290, 625
    - syntax of, 683
  - number sign. *See* #
  - numbers
    - decimal, 215
    - floating-point, 213
    - in regular expressions, 168
    - number of digits in, 137, 139, 144–145, 148–149
    - number of fractional digits in, 137, 145
  - numeric types, 213–220, 692–693
    - enumerating, 147–148
    - facets applicable to, 142, 148, 696
- 0**
- OASIS international consortium, 12
  - object-oriented concepts, 514–522
  - occurrence constraints
    - changing restrictiveness of, 321, 324–329, 618–620
    - for defaulted elements, 102
    - for replacement elements, 286
    - in all groups, 276, 310
    - in choice groups, 275
    - in element declarations, 94, 267, 322
    - in element references, 92, 267
    - in element wildcards, 602
    - in group references, 386, 390, 456
    - in sequence groups, 271
    - vs. DTDs, 477
  - open content, 292–298, 604–605**
    - adding/removing, 619**
    - and forward compatibility, 625**
    - default, 295–298**
    - extending, 311–312**
    - restricting, 329–331**
    - vs. other extension mechanisms, 600**
  - openContent element, 292, 311, 604
    - mode attribute of, 293, 311, 329–330
    - removing in restriction, 331**
    - syntax of, 292, 664
  - optional value
    - of use attribute, 283
    - of value attribute, 150
  - or operator (XPath 2.0), 356, 378**
  - ##other value (namespace attribute on wildcard), 287, 323, 602
  - override element, 459–463, 612–614**
    - annotations in, 581, 585**
    - order of new definitions in, 463**
    - syntax of, 463, 665**
    - top-level location of, 462**

- overrides, 33, 459–471, 612–614
  - and target namespace, 459–462, 572
  - of attribute groups, 467, 491–492
  - of complex types, 465
  - of global declarations, 466
  - of named model groups, 467
  - of simple types, 464–465
  - ripple effect of, 461–462
  - risks of, 468–471
  - vs. DTDs, 488
  - vs. other extension mechanisms, 600
- P**
- P**
  - in category escapes, 168
  - in durations, 229–233
- p element (HTML), 525
- \p multicharacter escape, 161, 169–170, 177
- \P multicharacter escape, 169–171
- parameter entities (DTDs)
  - external, 492
  - for attribute extensions, 490–491
  - for extensibility, 486–492
  - for reuse, 483–486
  - internal, 483–491
- parent keyword (XPath), 438
- parentheses. *See* ( )
- particles (of complex type), 262
- path expressions (XPath), 367–369, 435–440
  - default namespace for, 60
  - unprefixed names in, 440–441
- pattern facet, 137, 139, 148–149
  - changing restrictiveness of, 619–620
  - for built-in types, 695–698
  - for derived types, 140
  - for duration types, 231
  - for list types, 194
  - for numeric types, 219
  - for union types, 185
  - multiple occurrences of, 148
  - syntax of, 138, 665
- #PCDATA specifier (DTDs), 474–475, 478
- percent sign, in URIs, 251–252
- performance, 82
- period. *See* .
- Perl programming language, 159
- plus sign. *See* +
- position function (XPath 2.0), 364
- positiveInteger type, 217
  - comparing values of, 254–255
  - facets applicable to, 696
- prefixed names, 40
  - of attributes, 44–46, 120, 122
- prefixes, 28, 37, 98
  - in path expressions, 439
  - mapping to:
    - target namespace, 29, 53
    - XML Schema Namespace, 38, 50–52
  - naming rules for, 41
- prelexical facets, 151
- preprocess facet (Saxon), 155
- preserve value (whiteSpace facet), 104, 125, 151–152, 205, 254
- primitive types, 203
  - additional, 203
- processContents attribute
  - and forward compatibility, 623
  - lax value of, 288, 291, 602–603
    - in open content, 605
  - of attribute wildcard, 298, 315–316, 336–337, 602–603
  - of element wildcard, 287–289
  - skip value of, 287–289
  - strict value of, 288–289
  - syntax of, 684
- prohibited value
  - of use attribute, 283
  - of value attribute, 150
- proxy schemas, 74
- public attribute (notation element), 493
  - syntax of, 684

- punctuation signs, in category escapes, 168
- purchase orders, 503, 507
  - intermediate elements for, 527–531
- Q**
- QName type, 246
  - comparing values of, 254
  - facets applicable to, 247, 698
- qualified names (QNames), 40, 98–100, 246
  - local, 575, 580
  - local part of, 40
  - of attributes, 44–46, 122, 580
  - of elements, 40, 91
  - uniqueness of, 75–76, 91, 117
- qualified value
  - of `attributeFormDefault` attribute, 122
  - of `elementFormDefault` attribute, 99, 402, 576
  - of `form` attribute, 100, 122
- quantifiers (in regular expressions), 161, 165–166, 176–177
- question mark. *See* ?
- &quot; ; entity reference, 163
- R**
- `\r` single-character escape, 163, 165
- RDDL (Resource Directory Description Language), 590–592
- readability
  - and named/anonymous types, 134
  - and namespace prefixes, 37
  - of smaller schema documents, 557
- redefine element, 448–452
  - annotations in, 581, 585
  - syntax of, 451, 665
  - top-level location of, 451
- redefinition, 33, 448–459
  - and target namespace, 448, 450, 572
  - avoiding in web services, 548
  - of attribute groups, 456–458, 490–491
  - of complex types, 453–454
  - of named model groups, 454–456
  - of simple types, 452–453
  - ripple effect of, 451
  - risks of, 468–471
  - vs. DTDs, 486–487
- ref attribute
  - of attribute groups, 395
  - of attributes, 115, 117
  - of elements, 91, 267, 388
  - of identity constraints, 442**
  - of named model groups, 388–389
  - syntax of, 684
- refer attribute (`keyref` element), 430
  - syntax of, 684
- regular expressions, 148–149, 159–178
  - atoms in, 161–176
  - branches in, 159–161, 177–178
  - characters in, 161–162, 173
  - nested, 175
  - pieces in, 159, 161
  - quantifiers in, 161, 165–166, 176–177
  - ranges in, 173
- relational models, 503–514
- relationship elements, 512–514
- relationships, 507–514
  - many-to-many, 507–514
  - one-to-many, 507
  - one-to-one, 507
  - with references, 510–512
  - with relationship elements, 512–514
  - with repetition, 508–509
- RELAX NG schema language, 12–14
- rendition, 525–526
- replace value (`whiteSpace` facet), 151–152, 205
- replacement elements, 285–291
  - disallowed namespaces and names of, 289–291**
  - in derived element declarations, 323
  - namespaces of, 287
  - occurrence constraints for, 286
    - any number of, 293**
  - validating, 289
  - See also* element wildcards
- representation, 525

- required value
  - of use attribute, 283
  - of value attribute, 150, 234
- #REQUIRED specifier (DTDs), 483
- restriction, 303, 305
  - assertions in, 353–354, 371
  - of all groups:
    - in version 1.1, 625
  - of attribute wildcards, 458
  - of complex types, 455, 603
  - of content models, 455
  - of integer types, 142
  - of item types, 198
  - of list types, 136, 182, 190–194
  - of notations, 495
  - of simple types, 135–136, 138–140, 182
  - of union types, 185
  - with regular expressions, 148–149
- restriction element
  - base attribute of, 136
  - syntax for:
    - complex content, 319, 667
    - list type, 191, 666
    - simple content, 318, 667
    - simple type, 136, 666
    - union type, 186, 666
- restriction value
  - of block attribute, 344, 420
  - of final attribute, 153, 343, 419
- reusability, 8
  - and anonymous types, 134
  - of content models, 302, 385
  - of schemas, 543, 597
- reusable groups. *See* named model groups
- RFC 2396 standard, 251
- RFC 3066 standard, 211
- RFC 3548 standard, 249
- RFC 3987 standard, 251
- RFC 4646 standard, 211
- RFC 4647 standard, 211
- root element, 87, 95
  - specific for individual operations, 523
- round function (XPath 2.0), 358
- round-half-to-even function (XPath 2.0), 358
- Ruby annotations, 207
- Russian Doll design, 551–552
- S**
- S**
  - in category escapes, 169
  - in durations, 229–230, 232–233
  - \s and \S, multicharacter escapes, 166
  - Salami Slice design, 553–554
  - SAX (Simple API for XML), 635
  - saxon:preprocess facet, 155, 643, 646
  - schema documents, 57–77
    - combining, 61–77
    - comments in, 497–498
    - defaults of, 77
    - location of, 83, 630
    - missing external components of, 76
    - modularizing, 28–30, 57, 62, 492, 557–559, 565–574, 597
    - namespace declarations in, 52
    - natural language of, 59, 211
    - overriding, 461
    - readability of, 37, 134, 557
    - redefining, 448, 450
    - reusing, 544
    - structure of, 58–61
      - sections in, 585–586
    - top level of, 19
      - global declarations at, 89, 115
      - imports at, 68
      - includes at, 63
      - named model groups at, 386
      - overrides at, 462
      - redefines at, 451
      - version numbers of, 59, 628–629
        - fixed, 631
  - schema element, 28, 58–60, 393
    - annotations in, 581, 585
    - attributeFormDefault attribute of, 77, 122, 580
    - blockDefault attribute of, 77, 345

- schema element (*cont.*)
  - `defaultAttributes` attribute of, 399–400
  - `elementFormDefault` attribute of, 77, 99–101, 576, 578
  - `finalDefault` attribute of, 77, 153, 344, 419
  - syntax of, 59, 668
  - `targetNamespace` of, 49–50
  - `version` attribute of, 59, 626, 628–629, 631, 689
  - `xml:lang` attribute of, 59
  - `xpathDefaultNamespace` attribute of, 60, 373–375, 441
- schema languages, 9–14
- `schemaLocation` attribute
  - of `import` element, 68, 85
  - of `include` element, 63
  - syntax of, 685
- `schemaLocation` attribute (XSI namespace), 30, 51, 80, 83–87, 588
- of imported documents, 571
- schemas, 3–8
  - and namespaces, 48, 565
  - components of, 18–20
  - designing, 7–8, 502, 526–527, 539–580
  - documenting changes to, 637
  - extending, 8, 531, 542–543, 599–614
  - generating documentation from, 541, 584
  - interoperability of, 518
  - mapping to XHTML forms, 586
  - organizing, 527–537, 550–559
  - portability of, 156
  - preprocessing, 642
  - reusing, 8, 543–544, 557, 584, 596–597
  - tool compatibility of, 547
  - uses for, 5–7, 540–542
  - using with DTDs, 499
  - version numbers of, 626–634
  - versioning, 531, 547–549, 558–559
- Schematron schema language, 13–14
- selector element
  - syntax of, 433–434, 668
  - `xpath` attribute of, 435
  - `xpathDefaultNamespace` attribute of, 375
- semicolon, in regular expressions, 162
- separators, in category escapes, 169
- sequence group, 26, 270–272
  - extending, 486–488, 606–607
  - in complex content extensions, 307–308
  - in complex types, 390
  - in named model groups, 386
  - meaningless, 321
  - nesting, 275–276
  - occurrence constraints in, 271
  - overriding, 488
  - redefining, 486–488
  - restricting, 324–329
  - syntax of, 270, 668
  - turning into `all` or `choice` group, 619
  - vs. DTDs, 477
- service contracts, 541
- short type, 218
  - facets applicable to, 696
- simple content, 25, 262–263
  - assertions for, 354
  - default/fixed values for, 102
  - deriving complex types from, 303–304
  - extending, 305–306
  - restricting, 317–318
  - vs. database columns, 504
  - vs. DTDs, 475
- simple types, 19, 21, 23–25, 129–156
  - anonymous, 22, 96, 120–121, 132–134, 550
  - assertions for, 353–365
  - associating with element names, 96
  - base type of, 135–136, 139, 182
  - built-in. *See* built-in types
  - changing restrictiveness of, 619
  - deriving, 182
  - facets applicable to, 138–139

- global, 22
- implementation-defined**, 154, 642–645
- local, 19
- named, 120–121, 131–134, 550, 559–560, 597
- overriding**, 459, 464–465
  - risks of, 468–470
- patterns in, 160
- preventing derivation of, 152–153
- redefining, 449, 452–453, 600, 609
  - risks of, 468–470
- restricting, 24, 135–136, 138–140, 182
  - with regular expressions, 148–149
- reusing, 597
- turning into unions/lists, 619
- values of:
  - comparing, 253
  - default/fixed, 102
  - vs. complex types, 21
  - vs. datatypes, 201
  - vs. DTDs, 474
- `simpleContent` element, 263, 303–304, 306
  - syntax of, 304, 669
- `simpleType` element, 96, 120
  - final attribute of, 152–153
  - syntax of:
    - anonymous definition, 133, 669
    - named definition, 132, 669
    - using with `restriction`, 136
- single-character escapes, 165
- skip value (`processContents` attribute), 287–289
- smartphones, 524
- source attribute (`documentation` element), 582
  - syntax of, 685
- SOX (Schema for Object-oriented XML), 11
- space character
  - in regular expressions, 162–163, 166
  - in strings, 151, 205–206
- square brackets, in regular expressions, 160–162, 165, 171–172
- starts-with function (XPath 2.0)**, 357
- strict value (`processContents` attribute), 288–289
- string type, 97, 205–208
  - facets applicable to, 205–207, 695
  - values of:
    - comparing, 254–255
    - empty, 103, 143, 206
    - whitespace in, 104, 125, 151–152, 205–206, 254
- string types, 691–692
  - deriving list types from, 195
  - values of:
    - comparing, 220, 254
    - length of, 143–144
    - valid, 145
  - vs. integers, 220
  - whitespace in, 151–152
- string-length function (XPath 2.0)**, 357
- stylesheet element (XSLT), 631
- substitution groups, 32, 407–414, 607–609
  - and data binding tools, 518
  - compatibility of, 619
  - controlling, 418–420
  - declaring, 409–412
  - disadvantages of, 417
  - element declarations in, 95, 114
    - multiple**, 413–414
  - for inline elements, 525
  - head of, 408–409, 608
  - members of, 408–410
  - type constraints for, 412–413
  - vs. choice groups, 414–415
  - vs. DTDs, 489
  - vs. other extension mechanisms, 600
- substitution value
  - (`block/blockDefault` attributes), 420

- substitutionGroup attribute
    - (element element), 410–412
    - list of names in**, 413–414
    - syntax of, 685
  - substring, substring-after,
    - substring-before functions (XPath 2.0)**, 357
  - suffix value (mode attribute), 293, 296, 311, 329
  - sum function (XPath 2.0)**, 364
  - system attribute (notation element), 121, 493
    - syntax of, 686
- T**
- T
- in dates, 223
  - in durations, 229–230, 232–233
  - \t single-character escape, 163, 165
  - tab character
    - in regular expressions, 163, 165–166
    - in strings, 151, 205–206
  - tables of contents, for narrative content, 525
  - target namespace, 28–29, 48–54
    - and conditional type assignment**, 381
    - and global attributes, 117
    - and identity constraints, 54
    - and named model groups, 401
    - and notations, 494
    - changing, 619
    - for included documents, 62–66
    - in path expressions, 440–441
    - making default, 52
    - mapping prefixes to, 29, 53
    - multiple in assembled schema documents, 28–29
    - of overridden schema documents**, 459–462, 572
    - of redefined schema documents**, 448, 450, 572
  - targetNamespace attribute
    - of element or attribute elements**, 339–341
    - of schema element, 49–50
    - syntax of:
      - on local declaration**, 686
      - on schema, 686
  - ##targetNamespace value
    - of namespace attribute, 287
    - of notNamespace attribute**, 289
    - of xpathDefaultNamespace attribute**, 374
  - test attribute (alternative, assert, assertion elements), 352, 375–376
    - syntax of, 686
    - using XPath in**, 378–380
  - text () (XPath), 438
  - then keyword (XPath 2.0), 369–370
  - time type, 222–223
    - comparing values of, 255
    - facets applicable to, 697
  - time zones, 222–234
    - and bounds facets, 235
  - token type, 205–208
    - facets applicable to, 205–207, 695
    - values of:
      - comparing, 255
      - empty, 103, 143, 206
      - vs. integers, 220
      - whitespace in, 151–152, 205–206, 254
  - totalDigits facet, 137, 144–145
    - for built-in types, 695–698
    - syntax of, 138, 670
  - trading partners, 5
  - Trang schema converter, 10
  - true function (XPath 2.0)**, 358
  - type attribute
    - of alternative element**, 376
    - of attribute element, 120–121, 394
    - of element element, 96–97, 267, 387
    - syntax of, 687

**type attribute (XSI namespace)**, 51, 80, 120, 518, 600  
 avoiding in web services, 548  
 for member types, 187  
 for repeated element names, 415–416  
 for type derivation, 606  
 for type redefinition, 609  
 for type substitution, 342, 605  
 syntax of, 687  
**type constraints**, 412–413  
**type derivation**, 301–348, 605  
   by extension, 22, 27, 302–303, 403, 606  
   by restriction, 22, 301–303, 316–337  
   prohibiting, 343–348  
   vs. named model groups, 403–404, 520  
   vs. OO class inheritance, 514–518  
**type libraries**, 584  
**type substitution**, 115, 302, 341–342, 518, 605  
   avoiding in web services, 548  
   prohibiting, 344–346  
   vs. other extension mechanisms, 600  
**typeAvailable attribute (Version Control namespace)**, 644–645  
   **syntax of**, 687  
**types**, 4, 18–23  
   applicability of, 597  
   canonical representation of, 204  
   comparing, 432  
   definitions of, 18  
   extending, 516  
   hierarchy of, 22, 253  
   inheriting, 544  
   lexical representations of, 204, 254  
   named vs. anonymous, 22, 96  
   names of, 545, 597  
   redefining, 600, 609  
   reusing, 523–524, 544  
   target namespace of, 48  
   value spaces of, 204  
   *See also* complex, simple types

**typeUnavailable attribute (Version Control namespace)**, 644–645  
   **syntax of**, 688

## U

**unbounded value (maxOccurs attribute)**, 274  
   **in all groups**, 276  
**underscore**. *See* `_`  
**Unicode standard**, 162, 165, 167, 205, 251  
**union element**, 183–185  
   syntax of, 183, 670  
**union types**, 146, 181–187  
   comparing, 253  
   facets applicable to, 185, 698  
   lists of, 196  
   member types of, 182, 184–187  
   restricting, 185  
**union value (final/finalDefault attributes)**, 153  
**unique element**, 426–428  
   syntax of:  
     definition, 427, 670  
     reference, 442, 671  
**uniqueness constraints**, 423, 426–428  
   changing restrictiveness of, 322  
   defining, 425  
   fields in, 110, 426, 428  
   names of, 426  
   referencing, 442–444  
   scope of, 426, 428  
   selectors in, 426, 428  
**units of measurement**, 130  
**unparsed entities (DTDs)**, 240, 242, 493  
   and notations, 496  
**unprefixed names**, 40, 43  
   and default namespace, 40  
   of attributes, 44, 122  
   of elements, 39, 94  
**unqualified names**  
   and default namespace, 101  
   local, 98–101, 576–580  
   of attributes, 40, 122, 580

- unqualified names (*cont.*)
    - of elements, 40, 96
  - unqualified value
    - of `attributeFormDefault` attribute, 122
    - of `elementFormDefault` attribute, 77, 99, 101, 577–578
    - of `form` attribute, 100, 122
  - `unsignedByte`, `unsignedInt`, `unsignedLong`, `unsignedShort` types, 218
    - facets applicable to, 696
  - UPA (Unique Particle Attribution), 279, 602, 604, 624
  - upper-case function (XPath 2.0)**, 357
  - URIs (Uniform Resource Identifiers), 36, 250–251
  - URLs (Uniform Resource Locators), 36, 251
    - spaces in, 85
  - URNs (Uniform Resource Names), 36, 251
  - use attribute (attribute element), 117, 119, 394, 482–483
    - optional value of, 283
    - prohibited value of, 283
    - required value of, 283
    - syntax of, 688
  - user documentation. *See* documentation
  - UTC (Coordinated Universal Time), 233
- V**
- validation, 5, 81–82, 540–541
    - against both DTDs and schemas, 114, 499
    - and performance, 82
    - and specific root elements, 523
    - by type, 7, 21
    - choosing schema documents for, 30, 87
    - co-constraints for**, 586
    - of concatenated values, 130
    - of intra-document references, 525
    - strictness of, 287–289, 588
    - with RELAX NG, 12
    - with Schematron, 13–14
  - value attribute, 137
    - of `explicitTimezone` facet, 150, 234
    - syntax of, 688
  - `$value` variable, 353, 358
    - for list types**, 363–365
  - `vc:facetAvailable` attribute, 645–646
    - syntax of, 675
  - `vc:facetUnavailable` attribute, 645–646
    - syntax of, 675
  - `vc:maxVersion` attribute, 641
    - syntax of, 679
  - `vc:minVersion` attribute, 641
    - syntax of, 680
  - `vc:typeAvailable` attribute, 645
    - syntax of, 687
  - `vc:typeUnavailable` attribute, 645
    - syntax of, 688
  - Venetian Blind design, 554–557
  - version attribute (schema element), 59, 626, 628–629, 631
    - fixed value of, 631
    - syntax of, 689
  - Version Control Namespace (vc)**, 51, 641, 645–646
  - versioning, 617–639
    - and compatibility, 617–626
    - defining strategy for, 547–549, 636
    - granularity of, 558–559
    - intermediate elements for, 531
    - major and minor versions for, 626–627, 633
    - numbering, 626–634
  - vertical bar. *See* |
- W**
- `\w` and `\W`, multicharacter escapes, 167
  - W3C (World Wide Web Consortium), 11
  - web browsers, 524

- web services, 522–524
  - whitespace
    - in binary types, 249
    - in DTD processors, 474
    - in lists, 182, 188, 195
    - in regular expressions, 166
    - normalizing, 82, 254, 499
  - whiteSpace facet, 103, 137, 151
    - collapse value of, 104, 125, 151–152, 189, 205–206, 255
    - for built-in types, 695–698
    - for list types, 189
    - for NMTOKEN type, 243
    - for numeric types, 104, 125, 219
    - for strings, 104, 125, 205–207, 254
    - preserve value of, 104, 125, 151–152, 205, 254
    - replace value of, 151–152, 205
    - syntax of, 138, 671
  - wildcard escape character, 164
  - wildcards, 284–298, 600–604
    - adding/removing, 619
    - for attributes. *See* attribute wildcards
    - for forward compatibility, 623
    - location of, in complex type definitions, 602
    - negative**, 289–291, 625
    - restricting, 322–324
    - vs. DTDs, 480
    - vs. other extension mechanisms, 600
    - See also* attribute, element wildcards
  - wrapper lists, 531–532
  - WXS (W3C XML Schema). *See* XML Schema
- X**
- x-, in language names, 211
  - XDR (XML Data Reduced), 11
  - XHTML (Extensible HyperText Markup Language), 526
    - and simple types, 207
    - including elements from, 572
    - mapping schemas to forms in, 586
  - XLink (XML Linking Language)
    - for external documentation, 584
    - using attributes for, 115
  - XML DTD types, 236–246, 694
    - facets applicable to, 697
    - length of values of, 143–144
  - XML (Extensible Markup Language)
    - documents. *See* instances
    - intermediate elements in, 527–531, 546
    - names in, 167
    - separating content from representation in, 524–526
  - XML messages, 502–537
    - tightly coupling with databases, 504
  - XML Schema 1.0
    - all groups in, 532
    - attribute groups in, 394
    - element declarations vs. wildcards in, 604, 624
    - generic elements in, 535–537
    - redefinition in, 448–459
  - XML Schema 1.1, 11, 640–641
    - +INF value in, 213–214
    - all groups in, 276, 310–311, 391, 606, 625
    - alternative element in, 97, 375–376, 650
    - assertions in, 33, 137, 150, 185, 351–375, 651–652, 686, 695–698
    - attributes in Version Control Namespace in, 51, 641, 645–646
    - backward-compatible with 1.0, 11
    - conditional type assignment in, 375–383
    - date and time types in, 224–225, 231–233, 697
    - defaultAttributes and defaultAttributesApply attributes in, 284, 399–400, 674

- XML Schema 1.1 (*cont.*)
  - defaultOpenContent element in, 295–298, 655, 672, 681
  - element declarations in:
    - multiple, 413
    - vs. wildcards, 280, 624
  - element wildcards in, 625
  - elementFormDefault attribute in, 100
  - explicitTimezone facet in, 137–138, 150, 234, 657, 695–698
  - field element in, 375
  - final attribute in, 418
  - finalDefault attribute of, 153, 419
  - forward compatibility in, 625, 641–642, 679–680
  - ID type in, 236
  - implementation-defined facets in, 155, 642, 645–646, 675
  - implementation-defined simple types in, 154, 642–645, 687
  - inheritable attributes in, 126–127, 283, 382–383, 678
  - integer values in, 218
  - IRIs in, 251
  - namespaces in, 36–37, 43–44, 289–291, 459–462, 572, 683
  - open content in, 292–298, 311–312, 329–331, 600, 604–605, 619, 625, 664, 681
  - overrides in, 33, 459–471, 488, 491–492, 572, 581, 585, 600, 612–614, 665
  - primitive types in, 203
  - processContents attribute in, 605
  - referencing identity constraints in, 442–444, 660–661, 671
  - restrictions in, 320
  - selector element in, 375
  - substitution groups in, 413–414
  - targetNamespace attribute in, 339–341, 686
    - using XPath 2.0 with, 352, 355–365, 367–370, 378–380, 435–440
  - wildcards in, 289–291, 293, 683
    - xpathDefaultNamespace attribute in, 60, 373–375, 381, 441, 689
- XML Schema Instance Namespace, 51, 79–80, 108
- XML Schema Namespace, 50, 97
  - prefixes mapped to, 38, 50–52
- XML Schema recommendation, 11–14, 201
- xml:lang attribute, 59, 120, 211
  - syntax of, 678
- xmlns attribute, 39
- xmlns prefix, 37, 39
- xpath attribute, 435
  - syntax of, 689
- XPath language, 13
  - and list types, 190
  - attributes in, 367, 436–437
  - expressions in, 367–369, 435–440
  - processing relationships in, 511
  - unprefixed names in, 440–441
  - wildcards in, 436–437
- XPath 2.0 language
  - comparing types in, 359–362
  - conditional expressions in, 369–370
  - for assertions, 352
  - functions in, 357–359, 363–364
  - in conditional type assignment, 378–380
  - operators in, 355–356
- xpathDefaultNamespace attribute
  - ##defaultNamespace value of, 375
  - ##local value of, 375
  - ##targetNamespace value of, 374
  - of alternative element, 375, 381
  - of assert element, 375
  - of assertion element, 375
  - of field element, 375
  - of schema element, 60, 373–375, 441
  - of selector element, 375
  - syntax of, 689

- xs prefix, 28, 38, 50–52, 97
    - See also built-in types
  - xs:error built-in type, 380–381**
  - XSD (W3C XML Schema Definition Language). See XML Schema
  - xsd prefix, 38, 50–52
  - XSDL (XML Schema Definition Language). See XML Schema
  - xsi prefix, 80
  - xsi:nil attribute, 51, 80, 103, 107–110, 143
    - syntax of, 682
  - xsi:noNamespaceSchemaLocation attribute, 51, 80, 83–84, 86
    - syntax of, 683
  - xsi:schemaLocation attribute, 30, 51, 80, 83–87, 588
    - of imported documents, 571
    - syntax of, 685
  - xsi:type attribute, 51, 80, 120, 518, 600
    - avoiding in web services, 548
    - for member types, 187
    - for repeated element names, 415–416
    - for type derivation, 606
    - for type redefinition, 609
    - for type substitution, 342, 605
    - syntax of, 687
  - XSL-FO (Extensible Stylesheet Language Formatting Objects), 526
  - XSLT (Extensible Stylesheet Language Transformations), 635
    - and list types, 190
    - for upgrading instances, 639
    - processing messages in, 521, 532
    - schema-awareness of (version 2.0), 417
- Y**
- Y, in durations, 229–231
  - yearMonthDuration type, 231–232**
    - facets applicable to, 697
- Z**
- z
    - in category escapes, 169
    - in time values, 233–234
  - zero. See 0