

# Beginning XSLT and Xpath Transforming XML Documents and Data

Ian Williams



Updates, source code, and Wrox technical support at www.wrox.com

## **Beginning XSLT and XPath**

Introduction		xix
Chapter 1: First Steps with XSLT		.1
Chapter 2: Introducing XPath		25
Chapter 3: Templates, Variables, and Parameters		43
Chapter 4: Using Logic		61
Chapter 5: Sorting and Grouping		75
Chapter 6: Strings, Numbers, Dates, and Times		95
Chapter 7: Multiple Documents	1	15
Chapter 8: Processing Text		.41
Chapter 9: Identifiers and Keys	1	.59
Chapter 10: Debugging, Validation, and Documentation	1	.81
Chapter 11: A Case Study	2	01
Appendix A: Answers to Exercises	2	39
Appendix B: Extending XSLT	2	53
Appendix C: XSLT Processing Model	2	59
Appendix D: XSLT 2.0 Quick Reference	2	63
Appendix E: XSLT 2.0 Schema	3	15
Appendix F: XPath 2.0 Function Reference	3	41
Appendix G: References	3	77
Glossary	3	81
Index	3	85

## Beginning XSLT and XPath

#### **Transforming XML Documents and Data**

Ian Williams



Wiley Publishing, Inc.

## Beginning XSLT and XPath: Transforming XML Documents and Data

Published by Wiley Publishing, Inc. 10475 Crosspoint Boulevard Indianapolis, IN 46256 www.wiley.com

Copyright © 2009 by Ian Williams

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-47725-0

Manufactured in the United States of America

 $10\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1$ 

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

#### Library of Congress Control Number: 2009929458

**Trademarks**: Wiley, the Wiley logo, Wrox, the Wrox logo, Wrox Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

## **About the Author**

**Ian Williams** is an information designer specializing in XML technologies, and a software technical writer. He worked in the U.K. publishing industry before getting involved in information technology at OWL International, developers of the one of the first commercial hypertext products. Ian was a product manager there, and later a consultant working with large corporate customers.

Since 1998 Ian has worked on technical writing and information-design projects, most recently for Nokia, Reuters, and Volantis. He is co-author with Pierre Greborio of *Professional InfoPath 2003*, also from Wrox Press.

Ian lives with his wife, Helen, in Kent, in a converted lifeboat station overlooking the English Channel.

Executive Editor Carol Long

**Development Editor** Tom Dinse

**Technical Editor** Dan Squier

**Production Editor** Eric Charbonneau

**Copy Editor** Luann Rouff

**Editorial Director** Robyn B. Siesky

**Editorial Manager** Mary Beth Wakefield

#### Credits

**Production Manager** Tim Tate

Vice President and Executive Group Publisher Richard Swadley

Vice President and Executive Publisher Barry Pruett

Associate Publisher Jim Minatel

**Project Coordinator, Cover** Lynsey Stanford

**Proofreader** Candace English

Indexer Johnna VanHoose Dinse

Introduction	xix
Chapter 1: First Steps with XSLT	1
Transforming an XML Document to a Web Page	1
Using a Browser	2
Transforming Locally	11
Transforming XML Data to XML	14
Atom and RSS Elements	14
Developing the Stylesheet	17
Summary	23
Chapter 2: Introducing XPath	25
Nodes	25
Node Types	26
Node Properties	26
Data Model	27
Path Expressions	28
Using an XPath Analyzer	29
Axes	30
Node Tests	32
Predicates	33
Operators	33
XPath Functions	35
Strings	35
Dates, Times, and Durations	37
Nodes and Documents	38
Numbers	39
Summary	40
Exercises	40
Chapter 3: Templates, Variables, and Parameters	43
About Templates	44
Template Rules	44

Invoking a Rule	45
Using Modes	46
Setting Priorities	48
Built-in Rules	49
Named Templates	49
Variables	50
Parameters	55
Global Parameters	55
Template Parameters	57
Summarv	59
Exercises	60
Chapter 4: Using Logic	61
Conditional Processing	61
A Simple Choice	61
Multiple Choices	62
Using XPath for Conditional Tests	64
Iteration	64
Using Attribute Sets	64
Monitoring the Context	69
Processing XML Code	70
Summary	71
Exercises	72
Chapter 5: Sorting and Grouping	75
Sorting Content	75
Perform a Sort	81
Grouping	83
Common Values	84
Adjacent Items	85
Starting and Ending Conditions	91
Summary	93
Exercises	93
Chapter 6: Strings, Numbers, Dates, and Times	95
String Processing	95
About Collations	95
General Functions	96
Codepoints	96
Comparison	97

Concatenation	97
Simple Substrings	98
Using Regular Expressions	99
Normalizing Values	101
Escaping URIs	102
Numbers	103
Generating Numbers	103
Formatting Source Numbers	107
Dates and Times	109
Contextual Dates	109
Formatting	109
Combining and Converting Values	111
Durations	112
Time Zones	112
Summary	113
Exercises	114
Chapter 7: Multiple Documents	115
Modular Stylesheets	115
Including Modules	116
Imported Stylesheets	119
Source Documents	123
Using the document() Function	123
XPath Alternatives	125
Setting or Changing Context	127
Output Documents	127
Preparing a Feed Update	129
Splitting a Document	136
Summary	138
Exercises	139
Chapter 8: Processing Text	141
Controlling Whitespace	1/1
Stripping Space	1/2
Preserving Space	1/2
llsing /ysliteyt>	1/2
XMI to Text	143 1 <i>ЛЛ</i>
Taxt to XMI	1/6
Loading Unnarsed Text	140 146
Analyzing the Input	140
	140

	4 5 0
Alternatives to XSLI	153
XML Maps	154
XML Data in Excel	156
Summary	157
Exercises	158
Chapter 9: Identifiers and Keys	159
ID Datatypes	159
Using the id() Function	160
Keys	162
The key() Function	163
Generating Identifiers	165
Indexing Lines	165
Census to GEDCOM XML	170
Summary	179
Exercises	180
Chapter 10: Debugging, Validation, and Documentation	181
Debugging XSLT	181
Profiling	184
Verifying XHTML Output	185
Using Messages	186
Commenting Output	187
Using the error() Function	188
Type and Schema Validation	188
Types in XSLT	188
Using a Schema-Aware Processor	189
Documenting Your Stylesheets	195
Summary	199
Exercises	200
Chapter 11: A Case Study	201
Schema Overview	201
Common Elements and Attributes	202
Common Attributes	202
Block Elements	202
Inline Elements	203
The Quick-Reference Schema	204
Link Container Elements	205

Property Elements	206
Link Verification	208
Metadata Schemas	209
Resource Metadata	210
Subject Metadata	214
Reference Stylesheets	217
Link Module	220
Link Parameters	220
Function Module	223
Term Module	223
Term Parameters	225
Displaying Inline Terms	226
Building the Site	227
Generating the Reference Pages	228
Landing and Glossary Pages	233
Creating a Sitemap	234
Summary	237
Appendix A: Answers to Exercises	239
Chanter 1	239
Chapter 2	239
Ouestion 1	239
Question 2	240
Question 3	240
Chanter 3	241
Question 1	241
Question 2	242
Chanter 4	242
Ouestion 1	243
Question 2	243
Chapter 5	244
Ouestion 1	244
Ouestion 2	245
Ouestion 3	245
Chapter 6	246
Ouestion 1	246
Question 2	246
Question 3	247
Chapter 7	247
Question 1	247

Question 2	248
Question 3	248
Question 4	248
Chapter 8	249
Question 1	249
Question 2	249
Question 3	250
Chapter 9	250
Question 1	250
Question 2	250
Question 3	251
Chapter 10	251
Question 1	251
Question 2	251
Chapter 11	252
Appendix B: Extending XSLT	253
Stylesheet Functions	253
Calling an Extension Function	254
Function Libraries	255
EXSLT	255
FunctX	256
Vendor Extensions	256
User-Defined Extensions	257
Appendix C: XSLT Processing Model	259
The Data Model	259
Transforming	260
Parsing Inputs	260
Template Rules	261
Variables and Parameters	261
Controlling Processing	261
Outputs and Serialization	262
Appendix D: XSLT 2.0 Quick Reference	263
Elements	263
Attribute Groups	264
Types	264
Functions	264

XSLT Elements	264
xsl:analyze-string	264
xsl:apply-imports	265
xsl:apply-templates	265
xsl:attribute	266
xsl:attribute-set	267
xsl:call-template	268
xsl:character-map	269
xsl:choose	269
xsl:comment	270
xsl:copy	271
xsl:copy-of	271
xsl:decimal-format	272
xsl:declaration	274
xsl:document	274
xsl:element	274
xsl:fallback	275
xsl:for-each	276
xsl:for-each-group	276
xsl:function	277
xsl:if	278
xsl:import	279
xsl:import-schema	280
xsl:include	281
xsl:instruction	281
xsl:key	282
xsl:matching-substring	283
xsl:message	283
xsl:namespace	284
xsl:namespace-alias	285
xsl:next-match	285
xsl:non-matching-substring	286
xsl:number	286
xsl:otherwise	288
xsl:output	288
xsl:output-character	291
xsl:param	291
xsl:perform-sort	292
xsl:preserve-space	293
xsl:processing-instruction	294
xsl:result-document	294

xsl:sequence	297
xsl:sort	297
xsl:strip-space	299
xsl:stylesheet	299
xsl:template	300
xsl:text	301
xsl:transform	301
xsl:value-of	302
xsl:variable	303
xsl:with-param	304
Attribute Groups	304
Generic element attributes	304
Version attributes	305
Validation attributes	305
Types	306
XSLT Functions	306
current	307
current-group	307
current-grouping-key	307
document	307
element-available	308
format-date, format-dateTime, format-time,	308
format-number	309
function-available	310
generate-id	310
key	310
regex-group	311
system-property	311
type-available	312
unparsed-text, unparsed-text-available	312
unparsed-entity-public-id, unparsed-entity-uri	313
Appendix E: XSLT 2.0 Schema	315
W3C <sup>®</sup> Document License	336
Appendix F: XPath 2.0 Function Reference	341
Functions	341
abs	341
avg	342
adjust-date-to-timezone, adjust-dateTime-to-timezone, adjust-time-to-timezone	342
base-uri	342

boolean	343
ceiling	343
codepoint-equal	344
codepoints-to-string	344
collection	344
compare	345
concat	345
count	345
current-date, current-dateTime, current-time	346
data	346
dateTime	346
day-from-date, day-from-dateTime	347
days-from-duration	347
deep-equal	348
default-collation	348
distinct-values	348
doc, doc-available	349
document-uri	349
empty	349
encode-for-uri	350
ends-with	350
error	350
escape-html-uri	351
exactly-one	351
exists	352
false	352
floor	352
hours-from-dateTime, hours-from-time	353
id	353
idref	353
implicit-timezone	354
index-of	354
implicit-timezone	355
in-scope-prefixes	355
insert-before	355
iri-to-uri	356
lang	356
last	356
local-name	357
local-name-from-QName	357
lower-case	357
matches	358

max, min	358
minutes-from-dateTime, minutes-from-time	358
minutes-from-duration	359
month-from-date, month-from-dateTime	359
months-from-duration	360
month-from-date, month-from-dateTime	360
name	360
namespace-uri	361
namespace-uri-for-prefix	361
namespace-uri-from-QName	362
nilled	362
normalize-unicode	362
not	362
number	363
one-or-more	363
position	363
prefix-from-QName	364
QName	364
remove	364
replace	365
resolve-uri	365
resolve-QName	366
reverse	366
root	366
round	367
round-half-to-even	367
seconds-from-dateTime, seconds-from-time	368
seconds-from-duration	368
starts-with	368
static-base-uri	369
string	369
string-join	369
string-length	370
string-to-codepoints	370
subsequence	371
substring	371
substring-after	371
substring-before	372
sum	372
timezone-from-date, timezone-from-dateTime, timezone-from-time	373
tokenize	373
trace	374

translate	37/
	275
	373
unordered	375
upper-case	375
year-from-date, year-from-dateTime	376
years-from-duration	376
zero-or-one	376
Appendix G: References	377
Specifications	377
Tools and Resources	379
Glossary	381
Index	385

## Introduction

Welcome to XSLT and XPath, two members of the W3C XML family of standards. This book concentrates on using XSLT and XPath to solve problems that you are likely to encounter every day in writing XSLT stylesheets. I have tried to focus most attention on the features that you will need frequently, while still treating other aspects of the subject in brief. You can find additional detailed information in the Quick Reference appendixes, and in more advanced works such as Michael Kay's *XSLT 2.0 and XPath 2.0 Programmers Reference*, also in the Wrox list.

#### Who This Book Is For

I assume that you have a sound knowledge of XML and related web standards, such as XML Schema and XHTML. In an introductory book like this, there isn't enough space to fill in background information on these subjects.

Conversely, I don't assume that you are familiar with a particular programming language, or that you necessarily have a strong programming background. The chapters include a few comparisons with other languages, and as you'll see, XSLT takes a different approach from most of them.

If you are an experienced web author, or a technical writer who works regularly with XML, there is no reason why you can't pick up XSLT, leveraging your existing knowledge and skills. Quite a few practitioners that I know come from this kind of background.

This book aims to give you a good grounding in the basics of XSLT and XPath, concentrating on version 2.0 of both standards. It is *definitely not* aimed at experienced XSLT 1.0 programmers who require a skills update. Explaining the often-very-detailed differences between the two versions would simply confuse matters for beginners, who will do better to learn how to use techniques that are appropriate to the latest version.

### **XSLT in Outline**

According to the W3C specification, XSLT (Extensible Stylesheet Language: Transformations) is a "language for transforming XML documents into other XML documents." Given the widespread use of XML for exchanging data between applications and its success as a means of creating a wide range of document types, it is now easy to see why such a language is useful.

XSLT 1.0 was published as a recommendation late in 1999 shortly after XML 1.0, and at the time it didn't seem obvious that XSLT would become a success. It is probably so successful because it was, and still

is, most often used to generate HTML content for the web from XML sources — so much so that XSLT processing was incorporated into browser engines at an early date. XSLT 2.0 was published in January 2007 after a very long development period, which was complicated by some controversy and the need to track the development of XPath 2.0, on which is relies heavily.

The name Extensible Stylesheet Language: Transformations suggests that there is another "branch" to XSL — and there is: Extensible Stylesheet Language: Formatting Objects, or XSL-FO. XSL was initially part of a much more comprehensive project, covering both transformation and formatting semantics. The Formatting Objects recommendation (still formally titled "XSL") was published separately in October 2001. It is essentially an XML vocabulary used to specify the layout and properties of parts of printed pages, but I won't be covering it in this book.

#### XSLT Is Different

Writing code to handle XML transformations in XSLT differs markedly from the approach used in other programming languages. It is written in XML syntax in a declarative fashion, with processing specified in pattern-matching rules. By *declarative* I mean the opposite of the usual imperative approach; that is, an XSLT programmer does not define a sequence of actions, but specifies a number of rules that the result should satisfy.

XSLT has a type system based on XML Schema, and XPath expressions form an important second language, matching source document objects, selecting content for processing, and performing operations on content.

Compared to some other languages, it is much easier to learn the XSLT basics, but the different syntax and the nature of the XSLT processing model take some getting used to.

The XML source example that follows is written in the DocBook vocabulary (which is widely used in documenting information technology):

```
<?xml version="1.0" encoding="UTF-8"?>
<article>
    <title>A Simple Transform</title>
    <para>Because the transform is an XML document we need to start with an XML
    declaration, specifying the version and the encoding.</para>
    <para>The root element in a stylesheet is
        <emphasis>xsl:stylesheet</emphasis>, though the synonym
            <emphasis>xsl:transform</emphasis> may also be used. You must always
        specify the XSLT namespace, and it is important to set the version
        attribute correctly to match the type of processing required. In this book
        we generally specify version 2.0.</para>
```

<xsl:stylesheet

Shown next is a simple XSLT stylesheet that transforms the XML to HTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
  <xsl:template match="/">
     <html>
        <head>
           <title>
              <xsl:value-of select="title"/>
           </title>
        </head>
        <body>
           <xsl:apply-templates select="body"/>
        </body>
     </html>
  </xsl:template>
  <xsl:template match="title">
     <h1><xsl:value-of select="."/></h1>
  </xsl:template>
   <xsl:template match="para">
     <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="emphasis">
     <em><xsl:value-of select="."/></em>
  </xsl:template>
  <xsl:template match="programlisting">
     <xsl:value-of select="."/>
     </xsl:template>
</xsl:stylesheet>
```

The core HTML elements, highlighted in the preceding code example, form the output structure. They are written literally inside a series of sibling template instructions prefixed with xsl:.

The use of templates is similar to the non-HTML code you may have seen in web pages written using ASP, JSP, or PHP; but here the syntax is entirely XML, and the XSLT elements provide both the frame-work and the processing instructions.

#### Importance of XPath Language

XPath 2.0 is an expression language that is absolutely fundamental to XSLT 2.0 in several ways. An expression takes one or more input values and returns a value as output, so everywhere you can use a value you should also be able to use an expression to be evaluated. Usually, expressions are used as the value of attributes on XSLT elements — for example, <xsl:value-of select="a+b"/>.

XPath is used to match elements to XSLT template rules.

Another common use of XPath is selecting nodes in an XML document for subsequent processing. You can make a document-wide selection and refer to all the <list> elements, or be very specific by pointing to the class attribute in the first <para> in the third <section> of a document.

Then, you can use XPath to load documents, search strings, and manipulate numbers, using a very wide range of built-in functions.

#### Node Trees

XPath expressions operate on an abstract tree structure. Objects in the tree are nodes, of which there seven types, briefly described here:

- □ **Document:** The root of the tree representing an entire source document. I use the term *document node* to designate this node, to avoid confusing it with the root element of the source document.
- □ Element: Defined by pairs of start and end tags (e.g., <title></title>) or an empty element tag such as <img/> with no content.
- **Text:** A character sequence in an element.
- □ Attribute: The name and value of an attribute in an element start tag or an empty element tag, including all default value attributes in the schema.
- **Comment:** Comments in the XML source document, i.e., <!--->.
- **Processing instruction:** An instruction in the source document contained by <? ?>.
- □ **Namespace:** Namespace declaration copied to each element to which the declaration applies.

Figure I-1 shows how part of the tree of nodes for the DocBook article would look, with the document node outside of everything. Only the document, element, and text nodes for one instance of each element are shown. Each node contains the node type at the top, its name in the center in the case of elements, and the string value in the case of text nodes.





#### **Processing Overview**

The basic work of an XSLT processor is to use a stylesheet as a set of instructions for producing a result document from a source document. Generally, all three documents are XML documents, so XSLT is said to *transform* one input object to an output object of the same kind. Figure I-2 illustrates the process in outline.



Figure I-2

An XSLT processor treats the input and output documents as trees of nodes. You can think of these trees as being something like the W3C document object model, which some XSLT processors indeed use. However, unlike the DOM, there is no defined API in the XSLT specification. Different processors are free to implement this abstract data model in different ways.

The basic processing sequence comprises several steps, shown in Figure I-3.



Figure I-3

- **1.** The XML source document is parsed into a source tree.
- 2. The XSLT stylesheet is parsed to a stylesheet tree.
- **3.** A transform is applied to create a result tree.
- **4.** Serialization is applied to deliver content in the specified output format.

Essentially, the processor traverses the source tree in document order and looks for matching template rules in the stylesheet. If a match is found, then the instructions in the template are used to construct a node in the result tree. By default, the serialization creates an XML document, but specific instructions may be applied to output HTML, XHTML, or plain text.

The process can be more complex, potentially involving multiple sources, stylesheets and result trees, temporary trees held as variables, and multiple serializations. In Appendix B you'll find a more detailed view of how an XSLT processor works.

#### About the XSLT 2.0 Schema

As you work through the examples in the book, I'll introduce parts of the XSLT 2.0 schema so that you can examine the structure of the individual elements.

This schema is published separately by W3C, and it is not part of the XSLT 2.0 recommendation as such. The full schema is reproduced in Appendix E, and the latest version is also at www.w3.org/2007/schema-for-xslt20.xsd.

XSLT elements are broadly divided into two categories: declarations and instructions. For clarity, I mostly use one or the other of these terms, rather than call them elements.

The <xsl:declaration> and <xsl:instruction> are represented in the schema as abstract elements, which *never* appear in document instances, so you will not use them in a stylesheet; rather, you will use one of the elements in their substitution groups. A substitution group determines where the elements may appear. For example, you can see that <xsl:output> is a declaration from the substitutionGroup attribute value:

```
<xs:element name="output" substitutionGroup="xsl:declaration">
...
</xs:element>
```

This leads to a rather flat structure overall, with very little nesting of elements.

#### Declarations

*Declarations* define values such as the location of stylesheets to include or import, the method of output, global parameters, and the templates to use to match the source XML. These are top-level elements that immediately follow the root <xsl:stylesheet> element. They can appear in any order unless there is an <xsl:import> element, which must always appear first.

The schema declares the complex type xsl:generic-element-type with some common attributes:

```
<xs:complexType name="generic-element-type" mixed="true">
    <xs:attribute name="default-collation" type="xsl:uri-list"/>
    <xs:attribute name="exclude-result-prefixes" type="xsl:prefix-list-or-all"/>
    <xs:attribute name="extension-element-prefixes" type="xsl:prefix-list"/>
    <xs:attribute name="use-when" type="xsl:expression"/>
    <xs:attribute name="xpath-default-namespace" type="xs:anyURI"/>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

<xsl:declaration> is then defined as a generic-element-type, and the top-level elements are subsequently specified to be in the declaration substitution group:

<xs:element name="declaration" type="xsl:generic-element-type" abstract="true"/>

Here, for example, is the schema definition for the <xsl:output> element:

```
<xs:element name="output" substitutionGroup="xsl:declaration">
   <xs:complexType mixed="true">
      <xs:complexContent mixed="true">
         <xs:extension base="xsl:generic-element-type">
           <xs:attribute name="name" type="xsl:QName"/>
           <xs:attribute name="method" type="xsl:method"/>
           <xs:attribute name="byte-order-mark" type="xsl:yes-or-no"/>
           <xs:attribute name="cdata-section-elements" type="xsl:QNames"/>
           <xs:attribute name="doctype-public" type="xs:string"/>
           <xs:attribute name="doctype-system" type="xs:string"/>
           <xs:attribute name="encoding" type="xs:string"/>
           <xs:attribute name="escape-uri-attributes" type="xsl:yes-or-no"/>
           <xs:attribute name="include-content-type" type="xsl:yes-or-no"/>
           <xs:attribute name="indent" type="xsl:yes-or-no"/>
           <xs:attribute name="media-type" type="xs:string"/>
           <xs:attribute name="normalization-form" type="xs:NMTOKEN"/>
           <xs:attribute name="omit-xml-declaration" type="xsl:yes-or-no"/>
           <xs:attribute name="standalone" type="xsl:yes-or-no-or-omit"/>
           <xs:attribute name="undeclare-prefixes" type="xsl:yes-or-no"/>
           <xs:attribute name="use-character-maps" type="xsl:QNames"/>
           <xs:attribute name="version" type="xs:NMTOKEN"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
```

Figure I-4 shows a schema diagram for the declaration substitutions.

#### Instructions

Other XSLT elements known as *instructions* are used to specify the construction of result trees from individual elements and attributes in the source XML.

The xsl:versioned-element-type is defined as an extension of the generic-element-type, and followed by the instruction declaration. This is because every element except <xsl:output> may have a version attribute containing the XSLT version number, which may be used to indicate which version of XSLT the processor should apply:

<xs:element name="instruction" type="xsl:versioned-element-type" abstract="true"/>

<xsl:output> has an attribute with the same name, but this is intended to refer to the XML version specified in the output method.



#### Figure I-4

<xsl:value-of>, which you will meet in Chapter 1, is specified as an instruction. It is also has the type *sequence constructor*, which is a type that contains a series of XSLT instructions. The

xsl:sequence-constructor type includes elements that can contain a sequence constructor. The schema
extends the xsl:versioned-element-type, specifies the content model of a sequence-constructor
group, and defines the <xsl:value-of> instruction:

```
<xs:complexType name="sequence-constructor">
  <xs:complexContent mixed="true">
    <xs:extension base="xsl:versioned-element-type">
      <xs:group ref="xsl:sequence-constructor-group" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:group name="sequence-constructor-group">
  <xs:choice>
    <xs:element ref="xsl:variable"/>
    <xs:element ref="xsl:instruction"/>
    <xs:group ref="xsl:result-elements"/>
 </xs:choice>
</xs:group>
<xs:element name="value-of" substitutionGroup="xsl:instruction">
 <xs:complexType>
    <xs:complexContent mixed="true">
      <xs:extension base="xsl:sequence-constructor">
        <xs:attribute name="select" type="xsl:expression"/>
        <xs:attribute name="separator" type="xsl:avt"/>
        <xs:attribute name="disable-output-escaping" type="xsl:yes-or-no"</pre>
          default="no"/>
      </xs:extension>
    </xs:complexContent>
 </xs:complexType>
</xs:element>
```

The substitution diagram for instructions looks very similar to the one for declarations, but because of the number of instructions it is too large to include here.

## What You Need to Use This Book

I habitually use a limited set of tools and a single development environment: the open-source Eclipse IDE, and the edition of Oxygen XML Editor that goes with it. Now and then I'll refer to them in particular.

There is a wide and very useful range of XSLT processors and XML editors out there; and while I don't want to endorse one rather than another, there are some arguments for using Oxygen as you work through this book, even if only temporarily:

- □ Oxygen is a multiplatform Java application.
- □ Both the basic and schema-aware versions of the Saxon XSLT processor are bundled with it.