



Extensible Stylesheet Language (XSL)

Version 1.0

W3C Working Draft 18 October 2000

This version:

<http://www.w3.org/TR/2000/WD-xsl-20001018/>
(PDF by RenderX, XML file, HTML (one large file), ZIP file)

Latest version:

<http://www.w3.org/TR/xsl/>

Previous version:

<http://www.w3.org/TR/2000/WD-xsl-20000327>

Authors and Contributors:

Sharon Adler (IBM) <sca@us.ibm.com>
Anders Berglund (IBM) <alrb@us.ibm.com>
Jeff Caruso (Pageflex) <jcaruso@pageflexinc.com>
Stephen Deach (Adobe) <sdeach@adobe.com>
Paul Grosso (ArborText) <paul@arbortext.com>
Eduardo Gutentag (Sun) <eduardo.gutentag@eng.sun.com>
Alex Milowski (Lexica) <alex@milowski.com>
Scott Parnell (Xerox) <Scott.Parnell@usa.xerox.com>
Jeremy Richman (BroadVision) <Jeremy.Richman@BroadVision.com>
Steve Zilles (Adobe) <szilles@adobe.com>

Copyright © 2000 W3C (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

XSL is a language for expressing stylesheets. It consists of two parts:

1. a language for transforming XML documents, and
2. an XML vocabulary for specifying formatting semantics.

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.

Status of this document

This is a W3C Working Draft for review by W3C members and other interested parties. This working draft incorporates the proposed resolution of the issues raised during Last Call. The Working Group intends to submit a revised version of this specification for publication as a Candidate Recommendation in the near future. We are issuing this interim public draft as it sets out a number of changes made in response to comments received on the Last Call draft. Items under consideration for change for Candidate Recommendation include the name of the `font-height-override-before` and `font-height-override-after` properties. Please send detailed comments to xsl-editors@w3.org; [archives](#) of the comments are available. More general public discussion of XSL takes place on the [XSL-List](#) mailing list.

It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. The XSL Working Group will not allow early implementation to constrain its ability to make changes to this specification prior to final release. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than “work in progress”. A list of current W3C working drafts can be found at <http://www.w3.org/TR>.

This document has been produced as part of the [W3C Style Activity](#) by the [XSL Working Group](#) ([members only](#)).

Table of Contents

1. Introduction and Overview	1
1.1. Processing a Stylesheet	1
1.1.1. Tree Transformations	2
1.1.2. Formatting	3
1.2. Benefits of XSL	6
1.2.1. Paging and Scrolling	7
1.2.2. Selectors and Tree Construction	8
1.2.3. An Extended Page Layout Model	8
1.2.4. A Comprehensive Area Model	8
1.2.5. Internationalization and Writing-Modes	9
1.2.6. Linking	9
2. Introduction to XSL Transformation	10
2.1. Tree Construction	10
2.2. XSL Namespace	10
3. Introduction to Formatting	10
3.1. Conceptual Procedure	12
4. Area Model	13
4.1. Introduction	13
4.2. Rectangular Areas	15
4.2.1. Area Types	15
4.2.2. Common Traits	15
4.2.3. Geometric Definitions	16
4.2.4. Tree Ordering	18
4.2.5. Stacking Constraints	19
4.2.6. Font Baseline Tables	25
4.3. Spaces and Conditionality	25
4.3.1. Space-resolution Rules	26
4.4. Block-areas	26
4.4.1. Stacked Block-areas	27
4.4.2. Intrusion Adjustments	29
4.5. Line-areas	30
4.6. Inline-areas	31
4.6.1. Stacked Inline-areas	32
4.6.2. Glyph-areas	32
4.7. Ordering Constraints	32
4.7.1. General Ordering Constraints	32

4.7.2. Line-building	33
4.7.3. Inline-building	34
4.8. Keeps and Breaks	34
4.9. Rendering Model	35
4.9.1. Geometry	35
4.9.2. Viewport Geometry	36
4.9.3. Visibility	36
4.9.4. Border, Padding, and Background	36
4.9.5. Intrinsic Marks	36
4.9.6. Layering and Conflict of Marks	37
4.10. Sample Area Tree	38
4.11. List of Traits on Areas	38
5. Property Refinement / Resolution	39
5.1. Specified, Computed, and Actual Values, and Inheritance	40
5.1.1. Specified Values	40
5.1.2. Computed Values	40
5.1.3. Actual Values	41
5.1.4. Inheritance	41
5.2. Shorthand Expansion	41
5.3. Computing the Values of Corresponding Properties	42
5.3.1. Border and Padding Properties	42
5.3.2. Margin, Space, and Indent Properties	43
5.3.3. Height, and Width Properties	44
5.3.4. Overconstrained Geometry	46
5.4. Simple Property to Trait Mapping	46
5.4.1. Background-position-horizontal and background-position-vertical Properties	46
5.4.2. Column-number Property	46
5.4.3. Text-align Property	47
5.4.4. Text-align-last Property	47
5.4.5. z-index Property	47
5.5. Complex Property to Trait Mapping	47
5.5.1. Word-spacing and Letter-spacing Properties	47
5.5.2. Reference-orientation Property	47
5.5.3. Writing-mode and Direction Properties	47
5.5.4. Absolute-position Property	49
5.5.5. Relative-position Property	49
5.5.6. Text-decoration Property	49
5.5.7. Font Properties	49
5.6. Non-property Based Trait Generation	50
5.7. Property Based Transformations	50

5.7.1. Text-transform Property	50
5.8. Unicode Bidi Processing	50
5.9. Expressions	52
5.9.1. Property Context	53
5.9.2. Evaluation Order	53
5.9.3. Basics	53
5.9.4. Function Calls	54
5.9.5. Numerics	54
5.9.6. Absolute Numerics	55
5.9.7. Relative Numerics	55
5.9.7.1. Percents	56
5.9.7.2. Relative Lengths	56
5.9.8. Strings	56
5.9.9. Colors	56
5.9.10. Keywords	56
5.9.10.1. inherit	56
5.9.11. Lexical Structure	56
5.9.12. Expression Value Conversions	57
5.9.13. Definitions of Units of Measure	58
5.9.13.1. Pixels	58
5.10. Core Function Library	59
5.10.1. Number Functions	59
5.10.2. Color Functions	60
5.10.3. Font Functions	60
5.10.4. Property Value Functions	60
5.11. Property Datatypes	62
6. Formatting Objects	65
6.1. Introduction to Formatting Objects	65
6.1.1. Definitions Common to Many Formatting Objects	65
6.2. Formatting Object Content	66
6.3. Formatting Objects Summary	67
6.4. Declarations and Pagination and Layout Formatting Objects	72
6.4.1. Introduction	72
6.4.1.1. Page-sequence-masters	72
6.4.1.2. Page-masters	72
6.4.1.3. Page Generation	73
6.4.1.4. Flows and Flow Mapping	74
6.4.1.5. Constraints on Page Generation	74
6.4.1.6. Pagination Tree Structure	75
6.4.2. fo:root	75
6.4.3. fo:declarations	75
6.4.4. fo:color-profile	76

6.4.5. fo:page-sequence	76
6.4.6. fo:layout-master-set	78
6.4.7. fo:page-sequence-master	78
6.4.8. fo:single-page-master-reference	79
6.4.9. fo:repeatable-page-master-reference	79
6.4.10. fo:repeatable-page-master-alternatives	80
6.4.11. fo:conditional-page-master-reference	81
6.4.12. fo:simple-page-master	82
6.4.13. fo:region-body	86
6.4.14. fo:region-before	89
6.4.15. fo:region-after	90
6.4.16. fo:region-start	91
6.4.17. fo:region-end	93
6.4.18. fo:flow	94
6.4.19. fo:static-content	95
6.4.20. fo:title	95
6.5. Block-level Formatting Objects	96
6.5.1. Introduction	96
6.5.1.1. Example	96
6.5.1.1.1. Chapter and Section Titles, Paragraphs	96
6.5.2. fo:block	98
6.5.3. fo:block-container	99
6.6. Inline-level Formatting Objects	101
6.6.1. Introduction	101
6.6.1.1. Examples	101
6.6.1.1.1. First Line of Paragraph in Small-caps	101
6.6.1.1.2. Figure with a Photograph	101
6.6.1.1.3. Page numbering and page number reference	102
6.6.2. fo:bidi-override	104
6.6.3. fo:character	105
6.6.4. fo:initial-property-set	106
6.6.5. fo:external-graphic	107
6.6.6. fo:instream-foreign-object	108
6.6.7. fo:inline	110
6.6.8. fo:inline-container	111
6.6.9. fo:leader	113
6.6.10. fo:page-number	115
6.6.11. fo:page-number-citation	116
6.7. Formatting Objects for Tables	117
6.7.1. Introduction	117
6.7.1.1. Examples	117
6.7.1.1.1. Simple Table with CALS Column-width Specifications	117
6.7.2. fo:table-and-caption	121

6.7.3. fo:table	122
6.7.4. fo:table-column	124
6.7.5. fo:table-caption	125
6.7.6. fo:table-header	126
6.7.7. fo:table-footer	126
6.7.8. fo:table-body	127
6.7.9. fo:table-row	127
6.7.10. fo:table-cell	128
6.8. Formatting Objects for Lists	130
6.8.1. Introduction	130
6.8.1.1. Examples	131
6.8.1.1.1. Enumerated List	131
6.8.1.1.2. HTML-style "dl" lists	133
6.8.2. fo:list-block	138
6.8.3. fo:list-item	139
6.8.4. fo:list-item-body	139
6.8.5. fo:list-item-label	140
6.9. Dynamic Effects: Link and Multi Formatting Objects	140
6.9.1. Introduction	140
6.9.1.1. Examples	141
6.9.1.1.1. Expandable/Collapsible Table of Contents	141
6.9.1.1.2. Styling an XLink Based on the Active State	145
6.9.2. fo:basic-link	146
6.9.3. fo:multi-switch	147
6.9.4. fo:multi-case	148
6.9.5. fo:multi-toggle	149
6.9.6. fo:multi-properties	149
6.9.7. fo:multi-property-set	150
6.10. Out-of-Line Formatting Objects	151
6.10.1. Introduction	151
6.10.1.1. Floats	151
6.10.1.2. Footnotes	151
6.10.1.3. Conditional Sub-Regions	151
6.10.2. fo:float	152
6.10.3. fo:footnote	154
6.10.4. fo:footnote-body	155
6.11. Other Formatting Objects	155
6.11.1. Introduction	155
6.11.1.1. Example	155
6.11.2. fo:wrapper	156
6.11.3. fo:marker	157
6.11.4. fo:retrieve-marker	157
7. Formatting Properties	159

7.1. Description of Property Groups	159
7.2. XSL Areas and the CSS Box Model	161
7.3. Common Accessibility Properties	162
7.3.1. “source-document”	162
7.3.2. “role”	162
7.4. Common Absolute Position Properties	163
7.4.1. “absolute-position”	163
7.4.2. “top”	164
7.4.3. “right”	165
7.4.4. “bottom”	165
7.4.5. “left”	165
7.5. Common Aural Properties	166
7.5.1. “azimuth”	166
7.5.2. “cue-after”	167
7.5.3. “cue-before”	167
7.5.4. “elevation”	167
7.5.5. “pause-after”	168
7.5.6. “pause-before”	168
7.5.7. “pitch”	168
7.5.8. “pitch-range”	169
7.5.9. “play-during”	169
7.5.10. “richness”	169
7.5.11. “speak”	170
7.5.12. “speak-header”	170
7.5.13. “speak-numeral”	170
7.5.14. “speak-punctuation”	171
7.5.15. “speech-rate”	171
7.5.16. “stress”	171
7.5.17. “voice-family”	171
7.5.18. “volume”	172
7.6. Common Border, Padding, and Background Properties	172
7.6.1. “background-attachment”	172
7.6.2. “background-color”	173
7.6.3. “background-image”	173
7.6.4. “background-repeat”	174
7.6.5. “background-position-horizontal”	175
7.6.6. “background-position-vertical”	176
7.6.7. “border-before-color”	177
7.6.8. “border-before-style”	177
7.6.9. “border-before-width”	177
7.6.10. “border-after-color”	178
7.6.11. “border-after-style”	178

7.6.12. “border-after-width”	179
7.6.13. “border-start-color”	179
7.6.14. “border-start-style”	180
7.6.15. “border-start-width”	180
7.6.16. “border-end-color”	180
7.6.17. “border-end-style”	181
7.6.18. “border-end-width”	181
7.6.19. “border-top-color”	181
7.6.20. “border-top-style”	182
7.6.21. “border-top-width”	183
7.6.22. “border-bottom-color”	184
7.6.23. “border-bottom-style”	184
7.6.24. “border-bottom-width”	184
7.6.25. “border-left-color”	185
7.6.26. “border-left-style”	185
7.6.27. “border-left-width”	186
7.6.28. “border-right-color”	186
7.6.29. “border-right-style”	186
7.6.30. “border-right-width”	187
7.6.31. “padding-before”	187
7.6.32. “padding-after”	188
7.6.33. “padding-start”	188
7.6.34. “padding-end”	189
7.6.35. “padding-top”	189
7.6.36. “padding-bottom”	189
7.6.37. “padding-left”	190
7.6.38. “padding-right”	190
7.7. Common Font Properties	190
7.7.1. Fonts and Font Data	190
7.7.2. “font-family”	193
7.7.3. “font-selection-strategy”	194
7.7.4. “font-size”	195
7.7.5. “font-stretch”	197
7.7.6. “font-size-adjust”	198
7.7.7. “font-style”	199
7.7.8. “font-variant”	200
7.7.9. “font-weight”	201
7.8. Common Hyphenation Properties	202
7.8.1. “country”	202
7.8.2. “language”	202
7.8.3. “script”	203
7.8.4. “hyphenate”	204

7.8.5. “hyphenation-character”	204
7.8.6. “hyphenation-push-character-count”	205
7.8.7. “hyphenation-remain-character-count”	205
7.9. Common Margin Properties-Block	206
7.9.1. “margin-top”	206
7.9.2. “margin-bottom”	206
7.9.3. “margin-left”	207
7.9.4. “margin-right”	208
7.9.5. “space-before”	209
7.9.6. “space-after”	209
7.9.7. “start-indent”	210
7.9.8. “end-indent”	210
7.10. Common Margin Properties-Inline	210
7.10.1. “space-end”	211
7.10.2. “space-start”	211
7.11. Area Alignment Properties	212
7.11.1. “alignment-adjust”	221
7.11.2. “alignment-baseline”	223
7.11.3. “baseline-shift”	225
7.11.4. “display-align”	226
7.11.5. “dominant-baseline”	227
7.11.6. “relative-align”	229
7.12. Area Dimension Properties	230
7.12.1. “block-progression-dimension”	230
7.12.2. “content-height”	231
7.12.3. “content-width”	232
7.12.4. “height”	233
7.12.5. “inline-progression-dimension”	234
7.12.6. “max-height”	235
7.12.7. “max-width”	236
7.12.8. “min-height”	237
7.12.9. “min-width”	237
7.12.10. “scaling”	238
7.12.11. “scaling-method”	238
7.12.12. “width”	239
7.13. Block and Line-related Properties	240
7.13.1. “hyphenation-keep”	240
7.13.2. “hyphenation-ladder-count”	240
7.13.3. “last-line-end-indent”	241
7.13.4. “line-height”	241
7.13.5. “line-height-shift-adjustment”	243
7.13.6. “line-stacking-strategy”	243

7.13.7. “linefeed-treatment”	244
7.13.8. “space-treatment”	245
7.13.9. “text-align”	246
7.13.10. “text-align-last”	247
7.13.11. “text-indent”	248
7.13.12. “white-space-collapse”	249
7.13.13. “wrap-option”	250
7.14. Character Properties	250
7.14.1. “character”	250
7.14.2. “letter-spacing”	251
7.14.3. “suppress-at-line-break”	252
7.14.4. “text-decoration”	253
7.14.5. “text-shadow”	254
7.14.6. “text-transform”	255
7.14.7. “treat-as-word-space”	256
7.14.8. “word-spacing”	256
7.15. Color-related Properties	258
7.15.1. “color”	258
7.15.2. “color-profile-name”	258
7.15.3. “rendering-intent”	258
7.16. Float-related Properties	259
7.16.1. “clear”	259
7.16.2. “float”	262
7.17. Keeps and Breaks Properties	264
7.17.1. “break-after”	264
7.17.2. “break-before”	265
7.17.3. “keep-together”	265
7.17.4. “keep-with-next”	266
7.17.5. “keep-with-previous”	267
7.17.6. “orphans”	268
7.17.7. “widows”	268
7.18. Layout-related Properties	268
7.18.1. “clip”	268
7.18.2. “overflow”	269
7.18.3. “reference-orientation”	270
7.18.4. “relative-position”	272
7.18.5. “span”	272
7.19. Leader and Rule Properties	273
7.19.1. “leader-alignment”	273
7.19.2. “leader-pattern”	273
7.19.3. “leader-pattern-width”	274

7.19.4. “leader-length”	275
7.19.5. “rule-style”	275
7.19.6. “rule-thickness”	276
7.20. Properties for Dynamic Effects Formatting Objects	276
7.20.1. “active-state”	276
7.20.2. “auto-restore”	277
7.20.3. “case-name”	278
7.20.4. “case-title”	278
7.20.5. “destination-placement-offset”	279
7.20.6. “external-destination”	279
7.20.7. “indicate-destination”	280
7.20.8. “internal-destination”	280
7.20.9. “show-destination”	280
7.20.10. “starting-state”	281
7.20.11. “switch-to”	281
7.20.12. “target-presentation-context”	282
7.20.13. “target-processing-context”	283
7.20.14. “target-stylesheet”	284
7.21. Properties for Markers	284
7.21.1. “marker-class-name”	284
7.21.2. “retrieve-class-name”	285
7.21.3. “retrieve-position”	285
7.21.4. “retrieve-boundary”	286
7.22. Properties for Number to String Conversion	287
7.22.1. “format”	287
7.22.2. “grouping-separator”	287
7.22.3. “grouping-size”	287
7.22.4. “letter-value”	288
7.23. Pagination and Layout Properties	288
7.23.1. “blank-or-not-blank”	288
7.23.2. “column-count”	289
7.23.3. “column-gap”	289
7.23.4. “extent”	289
7.23.5. “flow-name”	290
7.23.6. “force-page-count”	290
7.23.7. “initial-page-number”	291
7.23.8. “master-name”	292
7.23.9. “maximum-repeats”	293
7.23.10. “odd-or-even”	294
7.23.11. “page-height”	294
7.23.12. “page-position”	295
7.23.13. “page-width”	296

7.23.14. “precedence”	296
7.23.15. “region-name”	297
7.24. Table Properties	298
7.24.1. “border-after-precedence”	298
7.24.2. “border-before-precedence”	298
7.24.3. “border-collapse”	299
7.24.4. “border-end-precedence”	299
7.24.5. “border-separation”	300
7.24.6. “border-start-precedence”	300
7.24.7. “caption-side”	300
7.24.8. “column-number”	302
7.24.9. “column-width”	302
7.24.10. “empty-cells”	303
7.24.11. “ends-row”	303
7.24.12. “number-columns-repeated”	304
7.24.13. “number-columns-spanned”	304
7.24.14. “number-rows-spanned”	305
7.24.15. “starts-row”	305
7.24.16. “table-layout”	305
7.24.17. “table-omit-footer-at-break”	306
7.24.18. “table-omit-header-at-break”	306
7.25. Writing-mode-related Properties	307
7.25.1. “direction”	312
7.25.2. “font-height-override-after”	313
7.25.3. “font-height-override-before”	313
7.25.4. “glyph-orientation-horizontal”	314
7.25.5. “glyph-orientation-vertical”	315
7.25.6. “unicode-bidi”	316
7.25.7. “writing-mode”	317
7.26. Miscellaneous Properties	319
7.26.1. “content-type”	319
7.26.2. “id”	320
7.26.3. “provisional-label-separation”	320
7.26.4. “provisional-distance-between-starts”	321
7.26.5. “ref-id”	321
7.26.6. “score-spaces”	321
7.26.7. “src”	322
7.26.8. “visibility”	322
7.26.9. “z-index”	323
7.27. Shorthand Properties	324
7.27.1. “background”	324
7.27.2. “background-position”	324

7.27.3. “border”	327
7.27.4. “border-bottom”	328
7.27.5. “border-color”	328
7.27.6. “border-left”	329
7.27.7. “border-right”	329
7.27.8. “border-style”	329
7.27.9. “border-spacing”	330
7.27.10. “border-top”	330
7.27.11. “border-width”	331
7.27.12. “cue”	331
7.27.13. “font”	332
7.27.14. “margin”	333
7.27.15. “padding”	334
7.27.16. “page-break-after”	334
7.27.17. “page-break-before”	335
7.27.18. “page-break-inside”	337
7.27.19. “pause”	338
7.27.20. “position”	338
7.27.21. “size”	339
7.27.22. “vertical-align”	341
7.27.23. “white-space”	344
7.27.24. “xml:lang”	345
8. Conformance	345
 Appendices	
A. Internationalization	346
A.1. Additional “writing-mode” values	346
B. Formatting Object Summary	349
B.1. Declaration and Pagination and Layout Formatting Objects	349
B.2. Block Formatting Objects	350
B.3. Inline Formatting Objects	350
B.4. Table Formatting Objects	351
B.5. List Formatting Objects	351
B.6. Link and Multi Formatting Objects	351
B.7. Out-of-line Formatting Objects	352
B.8. Other Formatting Objects	352
C. Property Summary	352
C.1. Explanation of Trait Mapping Values	352
C.2. Property Table: Part I	353

C.3. Property Table: Part II	365
D. References	378
D.1. Normative References	378
D.2. Other References	380
E. Property Index	381
F. Acknowledgements (Non-Normative)	388

This page is intentionally left blank.

1. Introduction and Overview

This specification defines the Extensible Stylesheet Language (XSL). XSL is a language for expressing stylesheets. Given a class of arbitrarily structured XML [W3C XML] documents or data files, designers use an XSL stylesheet to express their intentions about how that structured content should be presented; that is, how the source content should be styled, laid out, and paginated onto some presentation medium, such as a window in a Web browser or a hand-held device, or a set of physical pages in a catalog, report, pamphlet, or book.

1.1. Processing a Stylesheet

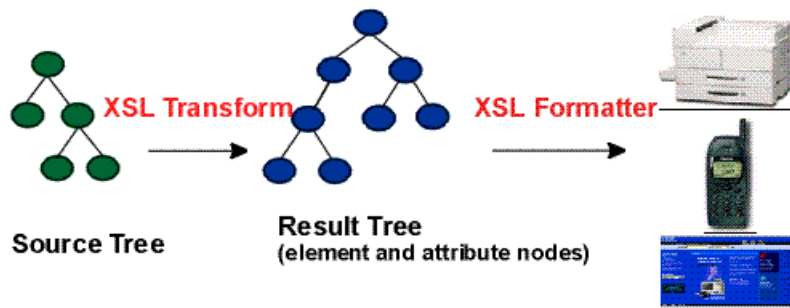
An XSL *stylesheet processor* accepts a document or data in XML and an XSL stylesheet and produces the presentation of that XML source content that was intended by the designer of that stylesheet. There are two aspects of this presentation process: first, constructing a result tree from the XML source tree and second, interpreting the result tree to produce formatted results suitable for presentation on a display, on paper, in speech, or onto other media. The first aspect is called *tree transformation* and the second is called *formatting*. The process of formatting is performed by the *formatter*. This formatter may simply be a rendering engine inside a browser.

Tree transformation allows the structure of the result tree to be significantly different from the structure of the source tree. For example, one could add a table-of-contents as a filtered selection of an original source document, or one could rearrange source data into a sorted tabular presentation. In constructing the result tree, the tree transformation process also adds the information necessary to format that result tree.

Formatting is enabled by including formatting semantics in the result tree. Formatting semantics are expressed in terms of a catalog of classes of *formatting objects*. The nodes of the result tree are formatting objects. The classes of formatting objects denote typographic abstractions such as page, paragraph, table, and so forth. Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling indents, word- and letter-spacing, and widow, orphan, and hyphenation control. In XSL, the classes of *formatting objects* and *formatting properties* provide the vocabulary for expressing presentation intent.

The XSL processing model is intended to be conceptual only. An implementation is not mandated to provide these as separate processes. Furthermore, implementations are free to process the source document in any way that produces the same result as if it were processed using the conceptual XSL processing model. A diagram depicting the detailed conceptual model is shown below.

XSL Two Processes: Transformation & Formatting



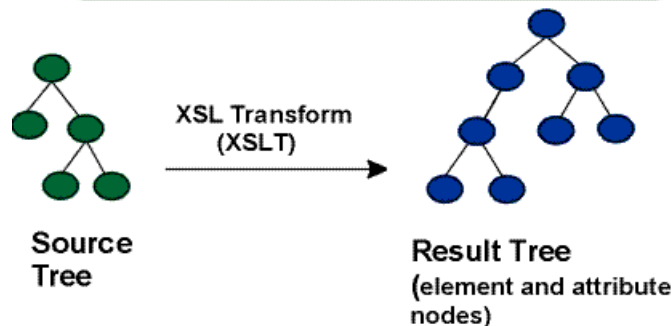
Result XML tree is the result of XSLT processing.

1.1.1. Tree Transformations

Tree transformation constructs the result tree. In XSL, this tree is called the *element and attribute tree*, with objects primarily in the "formatting object" namespace. In this tree, a formatting object is represented as an XML element, with the properties represented by a set of XML attribute-value pairs. The content of the formatting object is the content of the XML element. Tree transformation is defined in the XSLT Recommendation [XSLT]. A diagram depicting this conceptual process is shown below.

Transform to Another Vocabulary

With tree transformation, the structure of the result tree can be quite different from the structure of the source tree



In constructing the result tree, the source tree can be filtered and reordered, and arbitrary structure and generated content can be added.

The XSL stylesheet is used in tree transformation. A stylesheet contains a set of tree construction rules. The tree construction rules have two parts: a pattern that is matched against elements in the source tree and a template that constructs a portion of the result tree. This allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures.

In some implementations of XSL/XSLT, the result of tree construction can be output as an XML

document. This would allow an XML document which contains formatting objects and formatting properties to be output. This capability is neither necessary for an XSL processor nor is it encouraged. There are, however, cases where this is important, such as a server preparing input for a known client; for example, the way that a WAP (<http://www.wapforum.org/faqs/index.htm>) server prepares specialized input for a WAP capable hand held device. To preserve accessibility, designers of Web systems should not develop architectures that require (or use) the transmission of documents containing formatting objects and properties unless either the transmitter knows that the client can accept formatting objects and properties or the transmitted document contains a reference to the source document(s) used in the construction of the document with the formatting objects and properties.

1.1.2. Formatting

Formatting interprets the result tree in its formatting object tree form to produce the presentation intended by the designer of the stylesheet from which the XML element and attribute tree in the "fo" namespace was constructed.

The vocabulary of formatting objects supported by XSL - the set of `fo:` element types - represents the set of typographic abstractions available to the designer. Semantically, each formatting object represents a specification for a part of the pagination, layout, and styling information that will be applied to the content of that formatting object as a result of formatting the whole result tree. Each formatting object class represents a particular kind of formatting behavior. For example, the block formatting object class represents the breaking of the content of a paragraph into lines. Other parts of the specification may come from other formatting objects; for example, the formatting of a paragraph (block formatting object) depends on both the specification of properties on the block formatting object and the specification of the layout structure into which the block is placed by the formatter.

The properties associated with an instance of a formatting object control the formatting of that object. Some of the properties, for example "color", directly specify the formatted result. Other properties, for example 'space-before', only constrain the set of possible formatted results without specifying any particular formatted result. The formatter may make choices among other possible considerations such as esthetics.

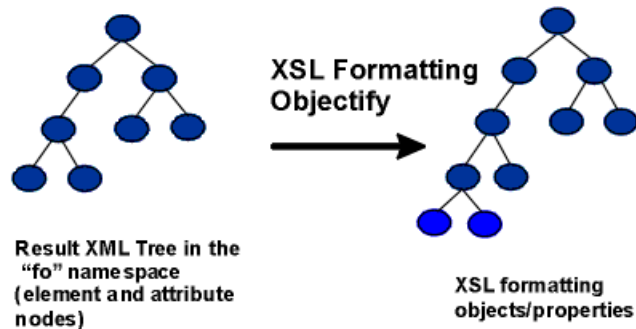
Formatting consists of the generation of a tree of geometric areas, called the *area tree*. The geometric areas are positioned on a sequence of one or more pages (a browser typically uses a single page). Each geometric area has a position on the page, a specification of what to display in that area and may have a background, padding, and borders. For example, formatting a single character generates an area sufficiently large enough to hold the glyph that is used to present the character visually and the glyph is what is displayed in this area. These areas may be nested. For example, the glyph may be positioned within a line, within a block, within a page.

Rendering takes the area tree, the abstract model of the presentation (in terms of pages and their collections of areas), and causes a presentation to appear on the relevant medium, such as a browser window on a computer display screen or sheets of paper. The semantics of rendering are not described in detail in this specification.

The first step in formatting is to "objectify" the element and attribute tree obtained via an XSLT transformation. Objectifying the tree basically consists of turning the elements in the tree into formatting object nodes and the attributes into property specifications. The result of this step is the *formatting object tree*.

Build the XSL Formatting Object Tree

The XSL FO tree is processed: characters are converted to character FOs and compound properties are built.



Some of the properties, for example "color", directly specify the formatted result.

Other properties, for example 'space-before', only constrain the set of possible formatted results without specifying any particular formatted result.

As part of the step of objectifying, the characters that occur in the result tree are replaced by `fo:character` nodes. Characters in text nodes which consist solely of whitespace characters and which are children of elements whose corresponding formatting objects do not permit `fo:character` nodes as children are ignored. Other characters within elements whose corresponding formatting objects do not permit `fo:character` nodes as children are errors.

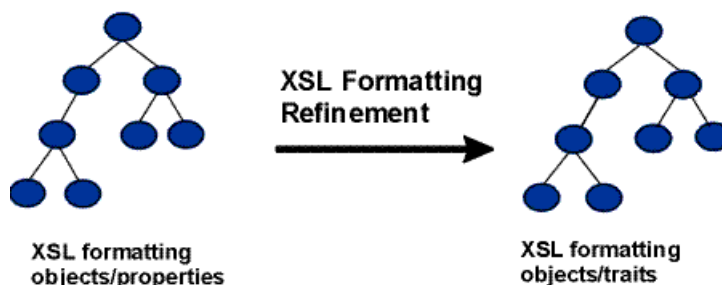
The content of the `fo:instream-foreign-object` is not objectified; instead the object representing the `fo:instream-foreign-object` element points to the appropriate node in the element and attribute tree. Similarly any non-XSL namespace child element of `fo:declarations` is not objectified; instead the object representing the `fo:declarations` element points to the appropriate node in the element and attribute tree.

The second phase in formatting is to refine the formatting object tree to produce the *refined formatting object tree*. The refinement process handles the mapping from properties to traits. This consists of: (1) shorthand expansion into individual properties, (2) mapping of corresponding properties, (3) determining computed values (may include expression evaluation), and (4) inheritance. Details on refinement are found in [§ 5 – Property Refinement / Resolution](#) on page 39.

The refinement step is depicted in the diagram below.

Refine the Formatting Object Tree

The XSL formatting object tree is refined in an iterative fashion.



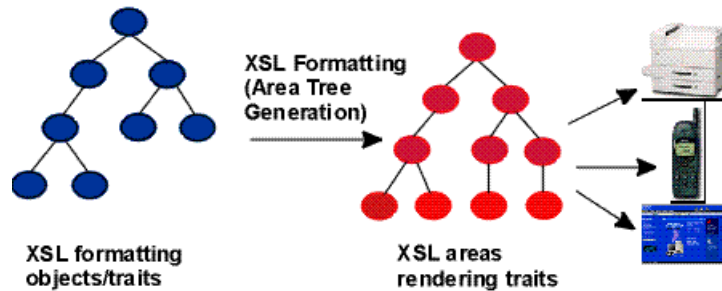
Property inheritance is resolved, computed values are processed, expressions are evaluated, and duplicate corresponding properties are removed.

The third step in formatting is the construction of the area tree. The area tree is generated as described in the semantics of each formatting object. The traits applicable to each formatting object class control how the areas are generated. Although every formatting property may be specified on every formatting object, for each formatting object class, only a subset of the formatting properties are used to determine the traits for objects of that class.

Area generation is depicted in the diagram below.

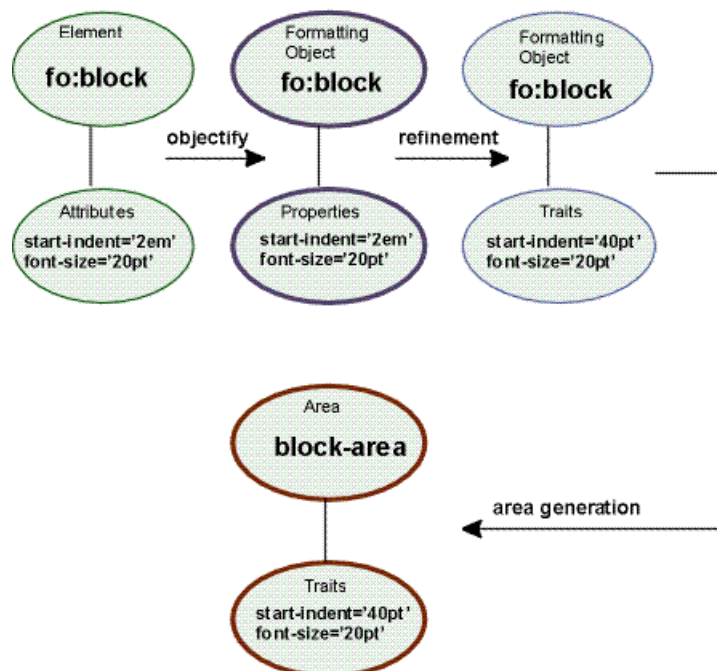
Generate the Area Tree

The last part of formatting describes the generation of a tree of geometric areas. These areas are positioned on a sequence of one or more pages.



Each geometric area has a position on the page, a specification of what to display in that area and may have a background, padding, and borders.

Summary of the Process



1.2. Benefits of XSL

Unlike the case of HTML, element names in XML have no intrinsic presentation semantics. Absent a

stylesheet, a processor could not possibly know how to render the content of an XML document other than as an undifferentiated string of characters. XSL provides a comprehensive model and a vocabulary for writing such stylesheets using XML syntax.

This document is intended for implementors of such XSL processors. Although it can be used as a reference manual for writers of XSL style sheets, it is not tutorial in nature.

XSL builds on the prior work on Cascading Style Sheets [CSS2] and the Document Style Semantics and Specification Language [DSSSL]. While many of XSL's formatting objects and properties correspond to the common set of properties, this would not be sufficient by itself to accomplish all the goals of XSL. In particular, XSL introduces a model for pagination and layout that extends what is currently available and that can in turn be extended, in a straightforward way, to page structures beyond the simple page models described in this specification.

1.2.1. Paging and Scrolling

Doing both scrollable document windows and pagination introduces new complexities to the styling (and pagination) of XML content. Because pagination introduces arbitrary boundaries (pages or regions on pages) on the content, concepts such as the control of spacing at page, region, and block boundaries become extremely important. There are also concepts related to adjusting the spaces between lines (to adjust the page vertically) and between words and letters (to justify the lines of text). These do not always arise with simple scrollable document windows, such as those found in today's browsers. However, there is a correspondence between a page with multiple regions, such as a body, header, footer, and left and right sidebars, and a Web presentation using "frames". The distribution of content into the regions is basically the same in both cases, and XSL handles both cases in an analogous fashion.

XSL was developed to give designers control over the features needed when documents are paginated as well as to provide an equivalent "frame" based structure for browsing on the Web. To achieve this control, XSL has extended the set of formatting objects and formatting properties. In addition, the selection of XML source components that can be styled (elements, attributes, text nodes, comments, and processing instructions) is based on XSLT and XPath, thus providing the user with an extremely powerful selection mechanism.

The design of the formatting objects and properties extensions was first inspired by DSSSL. The actual extensions, however, do not always look like the DSSSL constructs on which they were based. To either conform more closely with the CSS2 specification or to handle cases more simply than in DSSSL, some extensions have diverged from DSSSL.

There are several ways in which extensions were made. In some cases, it sufficed to add new values, as in the case of those added to reflect a variety of writing-modes, such as top-to-bottom and bottom-to-top, rather than just left-to-right and right-to-left.

In other cases, common properties that are expressed in CSS2 as one property with multiple simultaneous values, are split into several new properties to provide independent control over independent aspects of the property. For example, the "white-space" property was split into four properties: a "space-treatment" property that controls how white-space is processed, a "linefeed-treatment" property that controls how line-feeds are processed, a "white-space-collapse" property that controls how multiple consecutive spaces are collapsed, and a "wrap-option" property that controls whether lines are automatically wrapped when they encounter a boundary, such as the edge of a column. The effect of splitting a property into two or more (sub-)properties is to make the equivalent existing CSS2 property a "shorthand" for the set of sub-properties it subsumes.

In still other cases, it was necessary to create new properties. For example, there are a number of new

properties that control how hyphenation is done. These include identifying the script and country the text is from as well as such properties as "hyphenation-character" (which varies from script to script).

Some of the formatting objects and many of the properties in XSL come from the CSS2 specification, ensuring compatibility between the two.

There are four classes of XSL properties that can be identified as:

1. CSS properties by copy (unchanged from their CSS2 semantics)
2. CSS properties with extended values
3. CSS properties broken apart and/or extended
4. XSL-only properties

1.2.2. Selectors and Tree Construction

As mentioned above, XSL uses XSLT and XPath for tree construction and pattern selection, thus providing a high degree of control over how portions of the source content are presented, and what properties are associated with those content portions, even where mixed namespaces are involved.

For example, the patterns of XPath allow the selection of a portion of a string or the Nth text node in a paragraph. This allows users to have a rule that makes all third paragraphs in procedural steps appear in bold, for instance. In addition, properties can be associated with a content portion based on the numeric value of that content portion or attributes on the containing element. This allows one to have a style rule that makes negative values appear in "red" and positive values appear in "black". Also, text can be generated depending on a particular context in the source tree, or portions of the source tree may be presented multiple times with different styles.

1.2.3. An Extended Page Layout Model

There is a set of formatting objects in XSL to describe both the layout structure of a page or "frame" (how big is the body; are there multiple columns; are there headers, footers, or sidebars; how big are these) and the rules by which the XML source content is placed into these "containers".

The layout structure is defined in terms of one or more instances of a "simple-page-master" formatting object. This formatting object allows one to define independently filled regions for the body (with multiple columns), a header, a footer, and sidebars on a page. These simple-page-masters can be used in page sequences that specify in which order the various simple-page-masters shall be used. The page sequence also specifies how styled content is to fill those pages. This model allows one to specify a sequence of simple-page-masters for a book chapter where the page instances are automatically generated by the formatter or an explicit sequence of pages such as used in a magazine layout. Styled content is assigned to the various regions on a page by associating the name of the region with names attached to styled content in the result tree.

In addition to these layout formatting objects and properties, there are properties designed to provide the level of control over formatting that is typical of paginated documents. This includes control over hyphenation, and expanding the control over text that is kept with other text in the same line, column, or on the same page.

1.2.4. A Comprehensive Area Model

The extension of the properties and formatting objects, particularly in the area on control over the spacing of blocks, lines, and page regions and within lines, necessitated an extension of the CSS2 box

formatting model. This extended model is described in § 4 – [Area Model](#) on page 13 of this specification. The CSS2 box model is a subset of this model. See the mapping of the CSS2 box model terminology to the XSL Area Model terminology in § 7.2 – [XSL Areas and the CSS Box Model](#) on page 161. The area model provides a vocabulary for describing the relationships and space-adjustment between letters, words, lines, and blocks.

1.2.5. Internationalization and Writing-Modes

There are some scripts, in particular in the Far East, that are typically set with words proceeding from top-to-bottom and lines proceeding either from right-to-left (most common) or from left-to-right. Other directions are also used. Properties expressed in terms of a fixed, absolute frame of reference (using top, bottom, left, and right) and which apply only to a notion of words proceeding from left to right or right to left do not generalize well to text written in those scripts.

For this reason XSL (and before it DSSSL) uses a relative frame of reference for the formatting object and property descriptions. Just as the CSS2 frame of reference has four directions (top, bottom, left and right), so does the XSL relative frame of reference have four directions (before, after, start, and end), but these are relative to the "writing-mode". The "writing-mode" property is a way of controlling the directions needed by a formatter to correctly place glyphs, words, lines, blocks, etc. on the page or screen. The "writing-mode" expresses the basic directions noted above. There are writing-modes for "left-to-right - top-to-bottom" (denoted as "lr-tb"), "right-to-left - top-to-bottom" (denoted as "rl-tb"), "top-to-bottom - right-to-left" (denoted as "tb-rl") and more. See § 7.25.7 – [“writing-mode”](#) on page 317 for the description of the "writing-mode" property. Typically, the writing-mode value specifies two directions: the first is the inline-progression-direction which determines the direction in which words will be placed and the second is the block-progression-direction which determines the direction in which blocks (and lines) are placed one after another. In addition, the inline-progression-direction for a sequence of characters may be implicitly determined using bidirectional character types for those characters from the Unicode Character Database [[UNICODE Character Database](#)] for those characters and the Unicode Bidi Algorithm [[UNICODE TR9](#)].

Besides the directions that are explicit in the name of the value of the "writing-mode" property, the writing-mode determines other directions needed by the formatter, such as the shift-direction (used for sub- and super-scripts), etc.

1.2.6. Linking

Because XML, unlike HTML, has no built-in semantics, there is no built-in notion of a hypertext link. In this context, "link" refers to "hypertext link" as defined in <http://www.w3.org/TR/html401/struct/links.html#h-12.1> as well as some of the aspects of "link" as defined in <http://www.w3.org/TR/xlink/#intro>, where "link is a relationship between two or more resources or portions of resources, made explicit by an XLink linking element". Therefore, XSL has a formatting object that expresses the dual semantics of formatting the content of the link reference and the semantics of following the link.

NOTE: During the CR period the XSL WG and Linking WG will jointly develop additional examples and guidance on how to use these formatting objects given XPointer and XLink XML source.

XSL provides a few mechanisms for changing the presentation of a link target that is being visited. One of these mechanisms permits indicating the link target as such; another allows for control over the placement of the link target in the viewing area; still another permits some degree of control over the way the link target is displayed in relationship to the originating link anchor.

XSL also provides a general mechanism for changing the way elements are formatted depending on their active state. This is particularly useful in relation to links, to indicate whether a given link reference has already been visited, or to apply a given style depending on whether the mouse, for instance, is hovering over the link reference or not.

2. Introduction to XSL Transformation

2.1. Tree Construction

The Tree Construction is described in "XSL Transformations" [[XSLT](#)].

The provisions in "XSL Transformations" form an integral part of this recommendation and are considered normative.

2.2. XSL Namespace

The XSL namespace has the URI `http://www.w3.org/1999/XSL/Format`.

NOTE: The 1999 in the URI indicates the year in which the URI was allocated by the W3C. It does not indicate the version of XSL being used.

XSL processors must use the XML namespaces mechanism [[W3C XML Names](#)] to recognize elements and attributes from this namespace. Elements from the XSL namespace are recognized only in the stylesheet, not in the source document. Implementors must not extend the XSL namespace with additional elements or attributes. Instead, any extension must be in a separate namespace.

This specification uses the prefix `fo:` for referring to elements in the XSL namespace. However, XSL stylesheets are free to use any prefix, provided that there is a namespace declaration that binds the prefix to the URI of the XSL namespace.

An element from the XSL namespace may have any attribute not from the XSL namespace, provided that the expanded-name of the attribute has a non-null namespace URI. The presence of such attributes must not change the behavior of XSL elements and functions defined in this document. Thus, an XSL processor is always free to ignore such attributes, and must ignore such attributes without giving an error if it does not recognize the namespace URI. Such attributes can provide, for example, unique identifiers, optimization hints, or documentation.

It is an error for an element from the XSL namespace to have attributes with expanded-names that have null namespace URIs (i.e., attributes with unprefix names) other than attributes defined for the element in this document.

NOTE: The conventions used for the names of XSL elements, attributes, and functions are as follows: names are all lowercase, hyphens are used to separate words, dots are used to separate names for the components of complex datatypes, and abbreviations are used only if they already appear in the syntax of a related language such as XML or HTML.

3. Introduction to Formatting

The aim of this section is to describe the general process of formatting, enough to read the area model

and the formatting object descriptions and properties and to understand the process of refinement.

Formatting is the process of turning the result of an XSL transformation into a tangible form for the reader or listener. This process comprises several steps, some of which depend on others in a non-sequential way. Our model for formatting will be the construction of an *area tree*, which is an ordered tree containing geometric information for the placement of every glyph, shape, and image in the document, together with information embodying spacing constraints and other rendering information; this information is referred to under the rubric of *traits*, which are to areas what properties are to formatting objects and attributes are to XML elements. [§ 4 – Area Model](#) on page 13 will describe the area tree and define the default placement-constraints on stacked areas. However, this is an abstract model which need not be actually implemented in this way in a formatter, so long as the resulting tangible form obeys the implied constraints. Constraints might conflict to the point where it is impossible to satisfy them all. In that case, it is implementation-defined which constraints should be relaxed and in what order to satisfy the others.

Formatting objects are elements in the formatting object tree, whose names are from the XSL namespace; a formatting object belongs to a *class* of formatting objects identified by its element name. The formatting behavior of each class of formatting objects is described in terms of what areas are created by a formatting object of that class, how the traits of the areas are established, and how the areas are structured hierarchically with respect to areas created by other formatting objects. [§ 6 – Formatting Objects](#) on page 65 and [§ 7 – Formatting Properties](#) on page 159 describe formatting objects and their properties.

Some formatting objects are *block-level* and others are *inline-level*. This refers to the types of areas which they generate, which in turn refer to their default placement method. Inline-areas (for example, glyph-areas) are collected into lines and the direction in which they are stacked is the inline-progression-direction. Lines are a type of block-area and these are stacked in a direction perpendicular to the inline-progression-direction, called the block-progression-direction. See [§ 4 – Area Model](#) on page 13 for detailed descriptions of these area types and directions.

In Western writing systems, the block-progression-direction is "top-to-bottom" and the inline-progression-direction is "left-to-right". This specification treats other writing systems as well and introduces the terms "block" and "inline" instead of using absolute indicators like "vertical" and "horizontal". Similarly this specification tries to give relatively-specified directions ("before" and "after" in the block-progression-direction, "start" and "end" in the inline-progression-direction) where appropriate, either in addition to or in place of absolutely-specified directions such as "top", "bottom", "left", and "right". These are interpreted according to the value of the writing-mode property.

Central to this model of formatting is *refinement*. This is a computational process which finalizes the specification of properties based on the attribute values in the XML result tree. Though the XML result tree and the formatting object tree have very similar structure, it is helpful to think of them as separate conceptual entities. Refinement involves

- propagating the various inherited values of properties (both implicitly and those with an attribute value of "inherit"),
- evaluating expressions in property value specifications into actual values, which are then used to determine the value of the properties,
- converting relative numerics to absolute numerics,
- constructing some composite properties from more than one attribute

Some of these operations (particularly evaluating expressions) depend on knowledge of the area tree.

Thus refinement is not necessarily a straightforward, sequential procedure, but may involve look-ahead, back-tracking, or control-splicing with other processes in the formatter. Refinement is described more fully in § 5 – [Property Refinement / Resolution](#) on page 39.

To summarize, formatting proceeds by constructing an area tree (containing areas and their traits) which satisfies constraints based on information contained in the XML result tree (containing element nodes and their attributes). Conceptually, there are intermediate steps of constructing a formatting object tree (containing formatting objects and their properties) and refinement; these steps may proceed in an interleaved fashion during the construction of the area tree.

3.1. Conceptual Procedure

This subsection contains a conceptual description of how formatting could work. This conceptual procedure does not mandate any particular algorithms or data structures as long as the result obeys the implied constraints.

The procedure works by processing formatting objects. Each object, while being processed, may initiate processing in other objects. While the objects are hierarchically structured, the processing is not; processing of a given object is rather like a co-routine which may pass control to other processes, but pick up again later where it left off. The procedure starts by initiating the processing of the fo:root formatting object.

Unless otherwise specified, processing a formatting object creates areas and returns them to its parent to be placed in the area tree. Like a co-routine, it resumes control later and initiates formatting of its own children (if any), or some subset of them. The formatting object supplies parameters to its children based on the traits of areas already in the area tree, possibly including areas generated by the formatting object or its ancestors. It then disposes of the areas returned by its formatting object children. It might simply return such an area to its parent (and will always do this if it does not generate areas itself), or alternatively it might arrange the area in the area tree according to the semantics of the formatting object; this may involve changing its geometric position. It terminates processing when all its children have terminated processing (if initiated) and it is finished generating areas.

Some formatting objects do not themselves generate areas; instead these formatting objects simply return the areas returned to them by their children. Alternatively, a formatting object may continue to generate (and return) areas based on information discovered while formatting its own children; for example, the fo:page-sequence formatting object will continue generating pages as long as it contains a flow with unprocessed descendants.

Areas returned to an fo:root formatting object are page-viewport-areas, and are simply placed as children of the area tree root in the order in which they are returned, with no geometrical implications.

As a general rule, the order of the area tree parallels the order of the formatting object tree. That is, if one formatting object precedes another in the depth-first traversal of the formatting object tree, with neither containing the other, then all the areas generated by the first will precede all the areas generated by the second in the depth-first traversal of the area tree, unless otherwise specified. Typical exceptions to this rule would be things like side floats, before floats, and footnotes.

At the end of the procedure, the areas and their traits have been constructed, and they are required to satisfy constraints described in the definitions of their associated formatting objects, and in the area model section. In particular, size and position of the areas will be subject to the placement and spacing constraints described in the area model, unless the formatting object definition indicates otherwise.

The formatting object definitions, property descriptions, and area model are not algorithms. Thus, the

formatting object semantics do not specify how the line-breaking algorithm must work in collecting characters into words, positioning words within lines, shifting lines within a container, etc. Rather this specification assumes that the formatter has done these things and describes the constraints which the result is supposed to satisfy.

4. Area Model

In XSL, one creates a tree of formatting objects that serve as inputs or specifications to a formatter. The formatter generates a hierarchical arrangement of areas which comprise the formatted result. This section defines the general model of areas and how they interact. The purpose is to present an abstract framework which is used in describing the semantics of formatting objects. It should be seen as describing a series of constraints for conforming implementations, and not as prescribing particular algorithms.

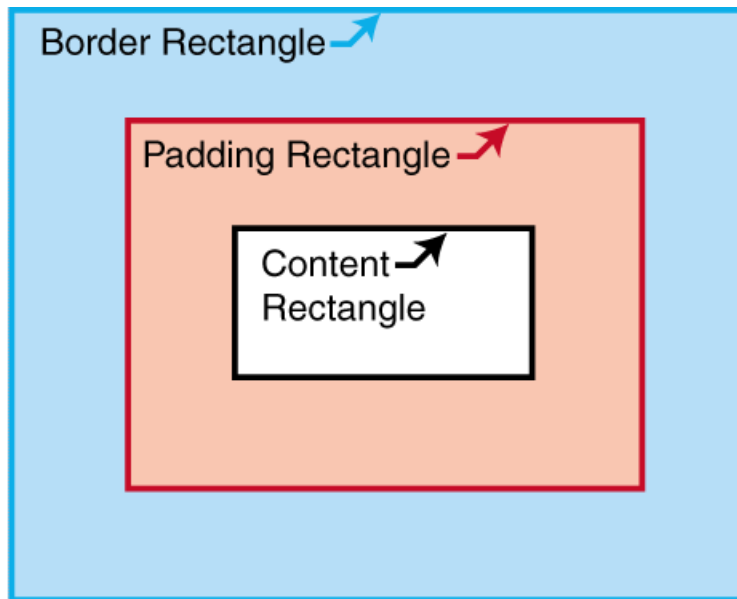
4.1. Introduction

The formatter generates an ordered tree, the *area tree*, which describes a geometric structuring of the output medium. The terms *child*, *sibling*, *parent*, *descendant*, and *ancestor* refer to this tree structure. The tree has a *root node*.

Each area tree node other than the root is called an *area* and is associated to a rectangular portion of the output medium. Areas are not formatting objects; rather, a formatting object generates zero or more rectangular areas, and normally each area is generated by a unique object in the formatting object tree.

NOTE: The only exceptions are when several leaf nodes of the formatting object tree are combined to generate a single area, for example when several characters in sequence generate a single ligature glyph. In all such cases, relevant properties such as *font-family* and *font-size* are the same for all the generating formatting objects (see section § 4.7.2 – [Line-building](#) on page 33).

An area has a *content-rectangle*, the portion in which its child areas are assigned, and optional *padding* and *border*. The diagram shows how these portions are related to one another. The outer bound of the border is called the *border-rectangle*, and the outer bound of the padding is called the *padding-rectangle*.



Each area has a set of *traits*, a mapping of names to values, in the way elements have attributes and formatting objects have properties. Individual traits are used either for rendering the area or for defining constraints on the result of formatting, or both. Traits used strictly for formatting purposes or for defining constraints may be called *formatting traits*, and traits used for rendering may be called *rendering traits*. Traits whose values are copied or derived from a property of the same or a corresponding name are listed in [Appendix C – Property Summary](#) on page 352 and [§ 5 – Property Refinement / Resolution](#) on page 39; other traits are listed in [§ 4.11 – List of Traits on Areas](#) on page 38.

NOTE: NOTE: traits are also associated with FOs during the process of refinement. Some traits are assigned during formatting, while others are already present after refinement.

The semantics of each type of formatting object that generates areas are given in terms of which areas it generates and their place in the area-tree hierarchy. This may be further modified by interactions between the various types of formatting objects. The properties of the formatting object determine what areas are generated and how the formatting object's content is distributed among them. (For example, a word that is not to be hyphenated may not have its glyphs distributed into areas on two separate line-areas.)

The traits of an area are either:

1. "directly-derived" -- The values of directly-derived traits are the computed value of a property of the same or a corresponding name on the generating formatting object, or
2. "indirectly-derived" -- The values of indirectly-derived traits are the result of a computation involving the computed values of one or more properties on the generating formatting object, other traits on this area or other interacting areas (ancestors, parent, siblings, and/or children) and/or one or more values constructed by the formatter. The calculation formula may depend on the type of the formatting object.

This description assumes that refined values have been computed for all properties of formatting objects in the result tree, i.e., all relative and corresponding values have been computed and the inheritable values have been propagated as described in [§ 5 – Property Refinement / Resolution](#) on page 39. This allows the process of inheritance to be described once and avoids a need to repeat information on computing values in this description.

4.2. Rectangular Areas

4.2.1. Area Types

There are two types of areas: *block-areas* and *inline-areas*. These differ according to how they are typically stacked by the formatter. An area can have block-area children or inline-area children as determined by the generating formatting object, but a given area's children must all be of one type. Although block-areas and inline-areas are typically stacked, some areas can be explicitly positioned.

A *line-area* is a special kind of block-area whose children are all inline-areas. A *glyph-area* is a special kind of inline-area which has no child areas, and has a single glyph image as its content.

Typical examples of areas are: a paragraph rendered by using an `fo:block` formatting object, which generates block-areas, and a character rendered by using an `fo:character` formatting object, which generates an inline-area (in fact, a *glyph-area*).

4.2.2. Common Traits

Associated with any area are two directions, which are derived from the generating formatting object's *writing-mode* and *reference-orientation* properties: the *block-progression-direction* is the direction for stacking block-area descendants of the area, and the *inline-progression-direction* is the direction for stacking inline-area descendants of the area. Another trait, the *shift-direction*, is present on inline-areas and refers to the direction in which baseline shifts are applied. Also the *glyph-orientation* defines the orientation of glyph-images in the rendered result.

The Boolean trait *is-reference-area* determines whether or not an area establishes a coordinate system for specifying indents. An area for which this trait is `true` is called a *reference-area*. Only a reference-area may have a block-progression-direction which is different from that of its parent. A reference-area may be either a block-area or an inline-area.

The Boolean trait *is-viewport-area* determines whether or not an area establishes an opening through which its descendant areas can be viewed, and can be used to present clipped or scrolled material; for example, in printing applications where bleed and trim is desired. An area for which this trait is `true` is called a *viewport-area*.

A common construct is a *viewport/reference pair*. This is a block-area viewport-area *V* and a block-area reference-area *R*, where *R* is the sole child of *V* and where the start-edge and end-edge of the content-rectangle of *R* are parallel to the start-edge and end-edge of the content-rectangle of *V*.

Each element has the traits *top-position*, *bottom-position*, *left-position*, and *right-position* which represent the distance from the edges of its content-rectangle to the like-named edges of the nearest ancestor reference-area (or the page-viewport-area in the case of areas generated by descendants of formatting objects whose absolute-position is `fixed`); the *left-offset* and *top-offset* determine the amount by which a relatively-positioned area is shifted for rendering. These traits receive their values during the formatting process, or in the case of absolutely positioned areas, during refinement.

The *block-progression-dimension* and *inline-progression-dimension* of an area represent the extent of the content-rectangle of that area in each of the two relative dimensions.

Other traits include:

- the *is-first* and *is-last* traits, which are Boolean traits indicating the order in which areas are generated and returned by a given formatting object. *is-first* is `true` for the first area (or only area) generated and returned by a formatting object, and *is-last* is `true` for the last area (or only area).

- the amount of space outside the border-rectangle: *space-before*, *space-after*, *space-start*, and *space-end* (though some of these may be required to be zero on certain classes of area);
- the thickness of each of the four sides of the padding: *padding-before*, *padding-after*, *padding-start*, and *padding-end*;
- the style, thickness, and color of each of the four sides of the border: *border-before*, etc.; and
- the background rendering of the area: *background-color*, *background-image*, and other background traits.

NOTE: "Before", "after", "start", and "end" refer to relative directions and are defined below.

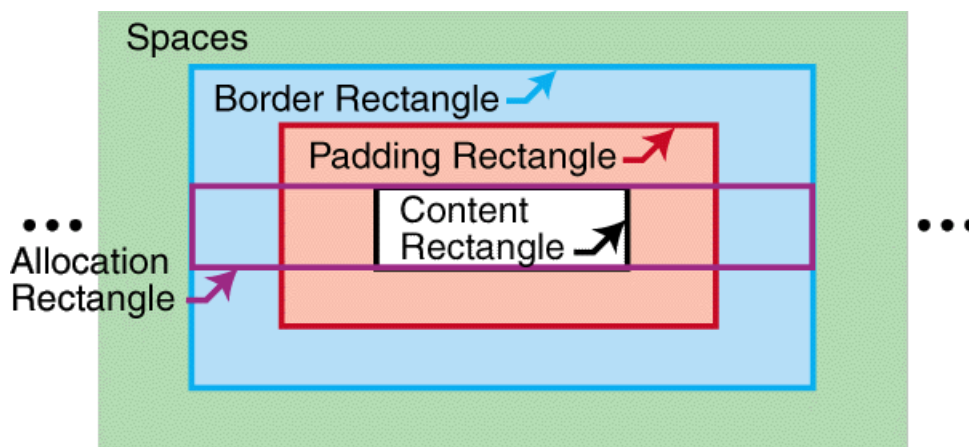
- a set of font traits (see [Common Font Properties](#) — § 7.7 on page 190) which are used to request a font that is deemed to be used within that area. The *nominal-font* for an area is determined by the font traits and the character descendants of the area. (see [Font Properties](#) — § 5.5.7 on page 49)

Unless otherwise specified, the traits of a formatting object are present on each of its generated areas, and with the same value. (However, see sections § 4.7.2 – [Line-building](#) on page 33 and § 4.9.4 – [Border, Padding, and Background](#) on page 36.)

4.2.3. Geometric Definitions

As described above, the *content-rectangle* is the rectangle bounding the inside of the padding and is used to describe the constraints on the positions of descendant areas. It is possible that marks from descendant glyphs or other areas may appear outside the content-rectangle.

Related to this is the *allocation-rectangle* of an area, which is used to describe the constraints on the position of the area within its parent area. For an inline-area this is either the *normal-allocation-rectangle* or the *expanded-allocation-rectangle*. The *normal-allocation-rectangle* extends to the content-rectangle in the block-progression-direction and to the border-rectangle in the inline-progression-direction. The *expanded-allocation-rectangle* extends outside the border-rectangle by an amount equal to the space-after in the block-progression-direction, an amount equal to the space-before in the opposite direction, an amount equal to the space-end in the inline-progression-direction, and an amount equal to the space-start in the opposite direction. Unless otherwise specified, the allocation-rectangle for an area is the normal-allocation-rectangle.

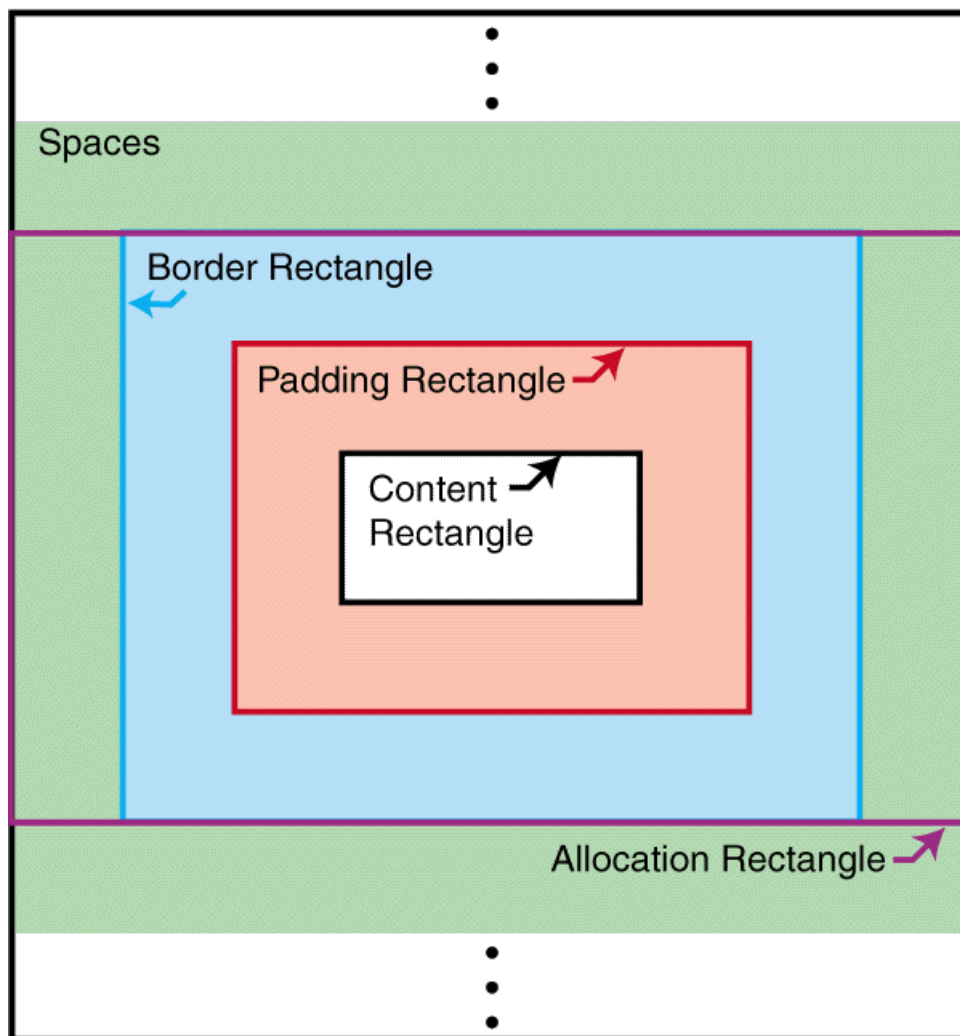


Allocation- and content-rectangles of an inline-area

For a block-area, the allocation-rectangle extends to the border-rectangle in the

block-progression-direction and outside the content-rectangle in the inline-progression-direction by an amount equal to the end-indent, and in the opposite direction by an amount equal to the start-indent.

NOTE: The inclusion of space outside the border-rectangle of a block-area in the inline-progression-direction does not affect placement constraints, and is intended to promote compatibility with the CSS box model.

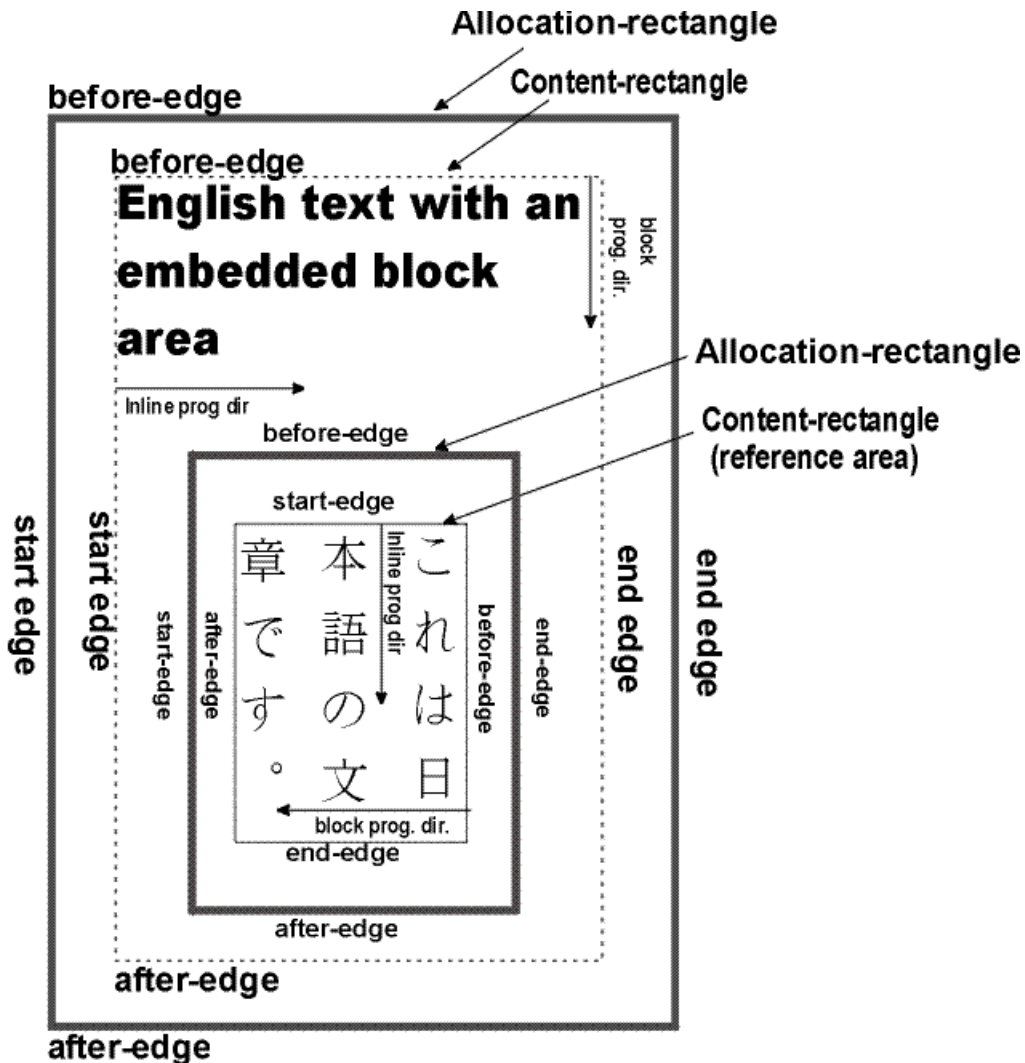


Allocation- and content-rectangles of a block-area

The edges of a rectangle are designated as follows:

- the *before-edge* is the edge occurring first in the block-progression-direction and perpendicular to it;
- the *after-edge* is the edge opposite the before-edge;
- the *start-edge* is the edge occurring first in the inline-progression-direction and perpendicular to it,
- the *end-edge* is the edge opposite the start-edge.

The following diagram shows the correspondence between the various edge names for a mixed writing-mode example:



For purposes of this definition, the content-rectangle of an area uses the inline-progression-direction and block-progression-direction of that area; but the border-rectangle, padding-rectangle, and allocation-rectangle use the directions of its parent area. Thus the edges designated for the content-rectangle may not correspond with the same-named edges on the padding-, border-, and allocation-rectangles. This is important in the case of nested block-areas with different writing-modes.

Each inline-area has a *alignment-point* determined by the formatter, on the start-edge of its allocation-rectangle; for a glyph-area, this is a point on the start-edge of the glyph on its alignment baseline (see below). This is script-dependent and does not necessarily correspond to the (0,0) coordinate point used for the data describing the glyph shape.

4.2.4. Tree Ordering

In the area tree, the set of areas with a given parent is ordered. The terms *initial*, *final*, *preceding*, and *following* refer to this ordering.

In any ordered tree, this sibling order extends to an ordering of the entire tree in at least two ways.

- In the *pre-order traversal order* of a tree, the children of each node (their order unchanged relative to one another) follow the node, but precede any following siblings of the node or of its ancestors.

- In the *post-order traversal order* of a tree, the children of each node precede the node, but follow any preceding siblings of the node or of its ancestors.

"Preceding" and "following", when applied to non-siblings, will depend on the extension order used, which must be specified. However, in either of these given orders, the leaves of the tree (nodes without children) are unambiguously ordered.

4.2.5. Stacking Constraints

This section defines the notion of *block-stacking constraints* and *inline-stacking constraints* involving areas. These are defined as ordered relations, i.e., if *A* and *B* have a stacking constraint it does not necessarily mean that *B* and *A* have a stacking constraint. These definitions are recursive in nature and some cases may depend upon simpler cases of the same definition. This is not circularity but rather a consequence of recursion. The intention of the definitions is to identify areas at any level of the tree which have only space between them.

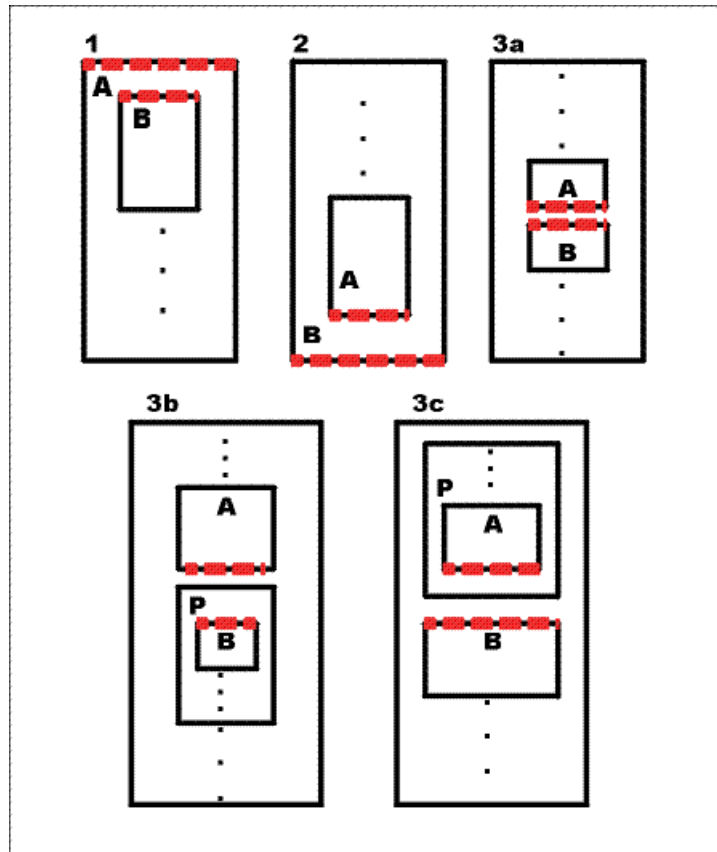
The *area-class* trait is an enumerated value which is `xsl-normal` for an area which is stacked with other areas in sequence. A *normal* area is an area for which this trait is `xsl-normal`. A *page-level-out-of-line* area is an area with area-class `xsl-footnote`, `xsl-before-float`, or `xsl-fixed`; placement of these areas is controlled by the `fo:page-sequence` ancestor of its generating formatting object. A *reference-level-out-of-line* area is an area with area-class `xsl-side-float` or `xsl-absolute`; placement of these areas is controlled by the formatting object generating the relevant reference-area. Areas with area-class equal to one of `xsl-normal`, `xsl-footnote`, or `xsl-before-float` are defined to be *stackable*, indicating that they are supposed to be properly stacked.

If *P* is a block-area, then there is a *fence before P* if *P* is a reference-area or if the border-before-width or padding-before-width of *P* are non-zero. Similarly, there is a *fence after P* if *P* is a reference-area or if the border-after-width or padding-after-width of *P* are non-zero.

If *A* and *B* are stackable areas, and *S* is a sequence of space-specifiers, it is defined that *A* and *B* have *block-stacking constraint S* if any of the following conditions holds:

1. *B* is a block-area which is the first normal child of *A*, and *S* is the sequence consisting of the space-before of *B*.
2. *A* is a block-area which is the last normal child of *B*, and *S* is the sequence consisting of the space-after of *A*.
3. *A* and *B* are both block-areas, and either
 - a. *B* is the next stackable sibling area of *A*, and *S* is the sequence consisting of the space-after of *A* and the space-before of *B*;
 - b. *B* is the first normal child of a block-area *P*, there is no fence before *P*, *A* and *P* have a block-stacking constraint *S'*, and *S* consists of *S'* followed by the space-before of *B*; or
 - c. *A* is the last normal child of a block-area *P*, there is no fence after *P*, *P* and *B* have a block-stacking constraint *S''*, and *S* consists of the space-after of *A* followed by *S''*.

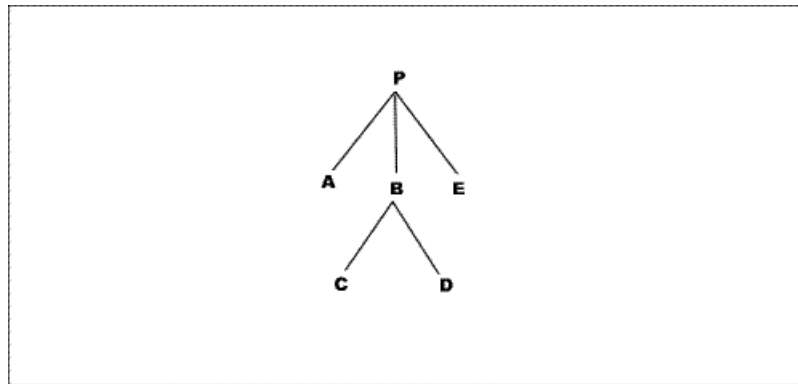
NOTE: The use of "stackable" in two places in the above definition allows block-stacking constraints to apply between areas of area-class `xsl-before-float` or `xsl-footnote`.



Adjacent Edges with Block-stacking

When *A* and *B* have a block-stacking constraint, the *adjacent edges* of *A* and *B* are an ordered pair recursively defined as:

- In case 1, the before-edge of the content-rectangle of *A* and the before-edge of the allocation-rectangle of *B*.
- In case 2, the after-edge of the content-rectangle of *A* and the after-edge of the allocation-rectangle of *B*.
- In case 3a, the after-edge of the allocation-rectangle of *A* and the before-edge of the allocation-rectangle of *B*.
- In case 3b, the first of the adjacent edges of *A* and *P*, and the before-edge of the allocation-rectangle of *B*.
- In case 3c, the after-edge of the allocation-rectangle of *A* and the second of the adjacent edges of *P* and *B*.

**Block-stacking constraint example**

Example. In this diagram each node represents a block-area. Assume that all padding and border widths are zero, and none of the areas are reference-areas. Then *P* and *A* have a block-stacking constraint, as do *A* and *B*, *A* and *C*, *B* and *C*, *C* and *D*, *D* and *B*, *B* and *E*, *D* and *E*, and *E* and *P*; these are the only pairs in the diagram having block-stacking constraints. If *B* had non-zero padding-after, then *D* and *E* would not have any block-stacking constraint (though *B* and *E* would continue to have a block-stacking constraint).

Inline-stacking constraints. This section will recursively define the inline-stacking constraints between two areas (either two inline-areas or one inline-area and one line-area), together with the notion of *fence before* and *fence after*; these definitions are interwoven with one another. This parallels the definition for block-stacking constraints, but with the additional complication that we may have a stacking constraint between inline-areas which are stacked in opposite inline-progression-directions. (This is not an issue for block-stacking constraints because a block-area which is not a reference-area may not have a block-progression-direction different from that of its parent.)

If *P* and *Q* have an inline-stacking constraint, then *P* has a *fence before Q* if *P* is a reference-area or has non-zero border-width or padding-width at the first adjacent edge of *P* and *Q*. Similarly, *Q* has a *fence after P* if *Q* is a reference-area or has non-zero border-width or padding-width at the second adjacent edge of *P* and *Q*.

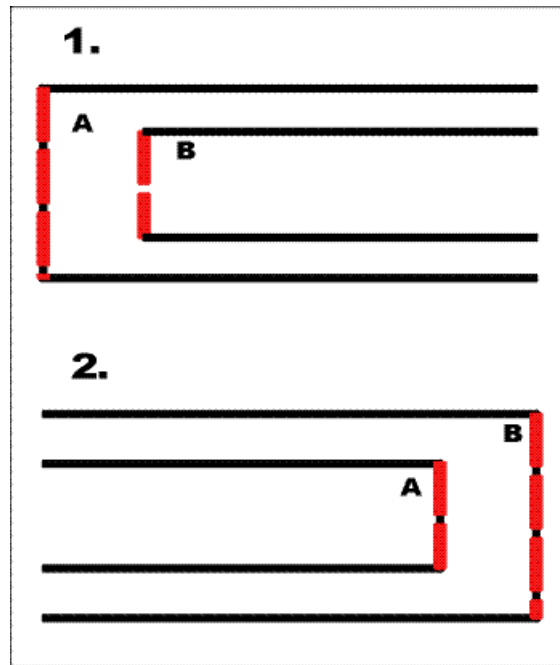
If *A* and *B* are normal areas, and *S* is a sequence of space-specifiers, it is defined that *A* and *B* have *inline-stacking constraint S* if any of the following conditions holds:

1. *A* is an inline-area or line-area, *B* is an inline-area which is the first normal child of *A*, and *S* is the sequence consisting of the space-start of *B*.
2. *B* is an inline-area or line-area, *A* is an inline-area which is the last normal child of *B*, and *S* is the sequence consisting of the space-end of *A*.
3. *A* and *B* are each either an inline-area or a line-area, and either
 - a. both *A* and *B* are inline-areas, *B* is the next normal sibling area of *A*, and *S* is the sequence consisting of the space-end of *A* and the space-start of *B*;
 - b. *B* is an inline-area which is the first normal child of an inline-area *P*, *P* has no fence after *A*, *A* and *P* have an inline-stacking constraint *S'*, the inline-progression-direction of *P* is the same as the inline-progression-direction of the nearest common ancestor area of *A* and *P*, and *S* consists of *S'* followed by the space-start of *B*.
 - c. *A* is an inline-area which is the last normal child of an inline-area *P*, *P* has no fence before *B*, *P* and *B* have an inline-stacking constraint *S''*, the inline-progression-direction of *P* is the same as the

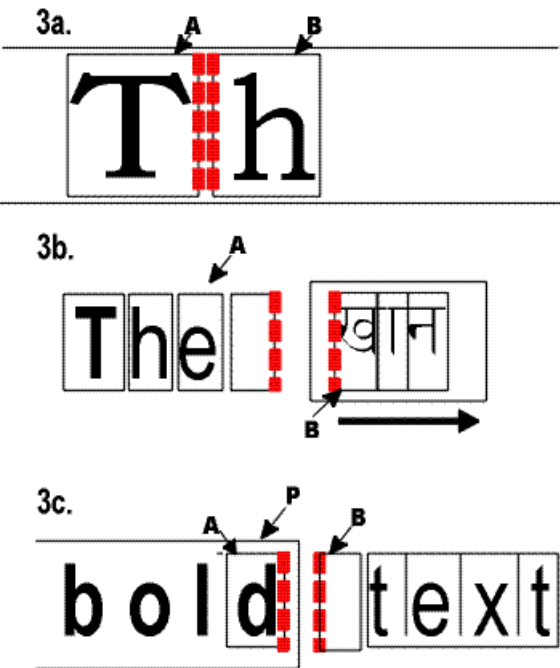
inline-progression-direction of the nearest common ancestor area of P and B , and S consists of the space-end of A followed by S'' .

d. B is an inline-area which is the last normal child of an inline-area P , P has no fence after A , A and P have an inline-stacking constraint S' , the inline-progression-direction of P is opposite to the inline-progression-direction of the nearest common ancestor area of A and P , and S consists of S' followed by the space-end of B .

e. A is an inline-area which is the first normal child of an inline-area P , P has no fence before B , P and B have an inline-stacking constraint S'' , the inline-progression-direction of P is opposite to the inline-progression-direction of the nearest common ancestor area of P and B , and S consists of the space-start of A followed by S'' .

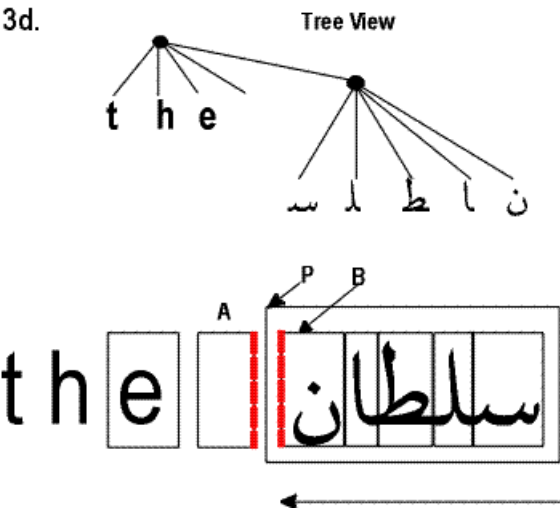


Adjacent Edges with Inline-stacking 1.

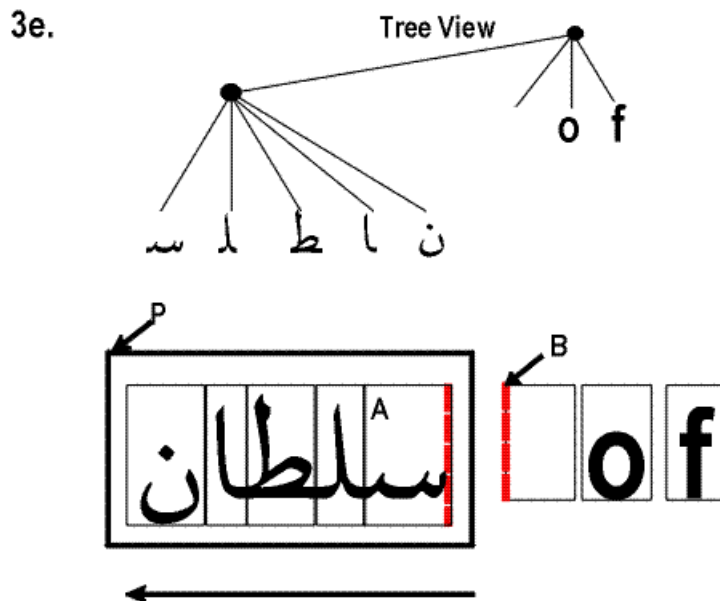


Adjacent Edges with Inline-stacking 2.

Mixed English and Arabic



Mixed English and Arabic



When *A* and *B* have an inline-stacking constraint, the *adjacent edges* of *A* and *B* are an ordered pair defined as:

- In case 1, the start-edge of the content-rectangle of *A* and the start-edge of the allocation-rectangle of *B*.
- In case 2, the end-edge of the content-rectangle of *A* and the end-edge of the allocation-rectangle of *B*.
- In case 3a, the end-edge of the allocation-rectangle of *A* and the start-edge of the allocation-rectangle of *B*.
- In case 3b, the first of the adjacent edges of *A* and *P*, and the start-edge of the allocation-rectangle of *B*.
- In case 3c, the end-edge of the allocation-rectangle of *A* and the second of the adjacent edges of *P* and *B*.
- In case 3d, the first of the adjacent edges of *A* and *P*, and the end-edge of the allocation-rectangle of *B*.
- In case 3e, the start-edge of the allocation-rectangle of *A* and the second of the adjacent edges of *P* and *B*.

Two areas are *adjacent* if they have a block-stacking constraint or an inline-stacking constraint. It follows from the definitions that areas of the same type (inline or block) can be adjacent only if all their non-common ancestors are also of the same type (up to but not including their nearest common ancestor). Thus, for example, two inline-areas which reside in different line-areas are never adjacent.

An area *A* *begins* an area *P* if *A* is a descendant of *P* and *P* and *A* have either a block-stacking constraint or an inline-stacking constraint. In this case the second of the adjacent edges of *P* and *A* is defined to be a *leading edge* in *P*. A space-specifier which applies to the leading edge is also defined to *begin P*.

Similarly, An area *A* *ends* an area *P* if *A* is a descendant of *P* and *A* and *P* have either a block-stacking constraint or an inline-stacking constraint. In this case the first of the adjacent edges of *A* and *P* is

defined to be a *trailing edge* in *P*. A space-specifier which applies to the trailing edge is also defined to *end P*

4.2.6. Font Baseline Tables

Each script has its preferred "baseline" for aligning glyphs from that script. Western scripts typically use a "alphabetic" baseline that touches at or near the bottom of capital letters. Further, for each font there is a preferred way of aligning embedded characters from different scripts, e.g., for a Western font there is a separate baseline for aligning embedded ideographic or Indic characters.

Each block-area and inline-area has a *dominant-baseline-identifier* trait whose value is a baseline identifier corresponding to the type of alignment expected for inline-area descendants of that area, and each inline-area has an *alignment-baseline* which specifies how the area is aligned to its parent. These traits are interpreted as described in section § 7.7.1 – [Fonts and Font Data](#) on page 190.

For each font, an *actual-baseline-table* maps these identifiers to points on the start-edge of the area. By abuse of terminology, the line in the inline-progression-direction through the point corresponding to the dominant-baseline-identifier is called the "dominant baseline."

The *text-altitude* of an area is defined in terms of the actual-baseline-table for the nominal-font of that area, and is normally a length equal to the distance between the dominant baseline and the text-before baseline. This is modified if the *font-height-override-before* has a value other than `use-font-metrics`. The *text-depth* is normally defined as a length equal to the distance between the dominant baseline and the text-after baseline. This is modified if the *font-height-override-after* has a value other than `use-font-metrics`.

4.3. Spaces and Conditionality

A space-specifier is a compound datatype whose components are minimum, optimum, maximum, conditionality and precedence.

Minimum, *optimum*, and *maximum* are lengths and can be used to define a constraint on a distance, namely that the distance should preferably be the optimum, and in any case no less than the minimum nor more than the maximum. Any of these values may be negative, which can (for example) cause areas to overlap, but in any case the minimum should be less than or equal to the optimum value, and the optimum less than or equal to the maximum value.

Conditionality is an enumerated value which controls whether a space-specifier has effect at the beginning or end of a reference-area or a line-area. Possible values are `retain` and `discard`; a *conditional* space-specifier is one for which this value is `discard`.

Precedence has a value which is either an integer or the special token `force`. A *forcing* space-specifier is one for which this value is `force`.

Space-specifiers occurring in sequence may interact with each other. The constraint imposed by a sequence of space-specifiers is computed by calculating for each space-specifier its associated *resolved space-specifier* in accordance with their conditionality and precedence, as shown below in the space-resolution rules.

The constraint imposed on a distance by a sequence of resolved space-specifiers is additive; that is, the distance is constrained to be no less than the sum of the resolved minimum values and no larger than the sum of the resolved maximum values.

4.3.1. Space-resolution Rules

To compute the resolved space-specifier of a given space-specifier *S*, consider the maximal inline-stacking constraint or block-stacking constraint containing *S*. The resolved space-specifier of *S* is a non-conditional, forcing space-specifier computed in terms of this sequence.

1. If any of the space-specifiers (in the maximal sequence) is conditional, and begins a reference-area or line-area, then it is *suppressed*, which means that its resolved space-specifier is zero. Further, any conditional space-specifiers which consecutively follow it in the sequence are also suppressed.

If a conditional space-specifier ends a reference-area or line-area, then it is suppressed together with any other conditional space-specifiers which consecutively precede it in the sequence.

2. If any of the remaining space-specifiers is forcing, all non-forcing space-specifiers are suppressed, and the value of each of the forcing space-specifiers is taken as its resolved value.
3. Alternatively if all of the remaining space-specifiers are non-forcing, then the resolved space-specifier is defined in terms of those space-specifiers whose precedence is numerically highest, and among these those whose optimum value is the greatest. All other space-specifiers are suppressed. If there is only one of these then its value is taken as its resolved value.

Otherwise, when there are two or more space-specifiers all of the same highest precedence and the same (largest) optimum, the resolved space-specifier of the last space-specifier in the sequence is derived from these spaces by taking their common optimum value as its optimum, the greatest of their minimum values as its minimum, and the least of their maximum values as its maximum, and all other space-specifiers are suppressed.

Example. Suppose the sequence of space values occurring at the beginning of a reference-area is: first, a space with value 10 points (that is minimum, optimum, and maximum all equal to 10 points) and conditionality `discard`; second, a space with value 4 points and conditionality `retain`; and third, a space with value 5 points and conditionality `discard`; all three spaces having precedence zero. Then the first (10 point) space is suppressed under rule 1, and the second (4 point) space is suppressed under rule 3. The resolved value of the third space is a non-conditional 5 points, even though it originally came from a conditional space.

The padding of a block-area does not interact with any space-specifier (except that by definition, the presence of padding at the before- or after-edge prevents areas on either side of it from having a stacking constraint.)

The border or padding at the before-edge or after-edge of a block-area may be specified as conditional. If so, then it is set to zero if its associated edge is a leading or trailing edge in a reference-area. In this case, the border or padding is taken to be zero for purposes of the stacking constraint definitions.

4.4. Block-areas

Block-areas have several traits which typically affect the placement of their children. The *line-height* is used in line placement calculations. The *line-stacking-strategy* trait controls what kind of allocation is used for descendant line-areas and has an enumerated value (either `font-height`, `max-height`, or `line-height`). This is all rigorously described below. All areas have these traits, but they only have relevance for areas which have stacked line-area children.

The *space-before* and *space-after* traits determine the distance between the block-area and surrounding block-areas.

A block-area which is not a line-area typically has its size in the inline-progression-direction determined

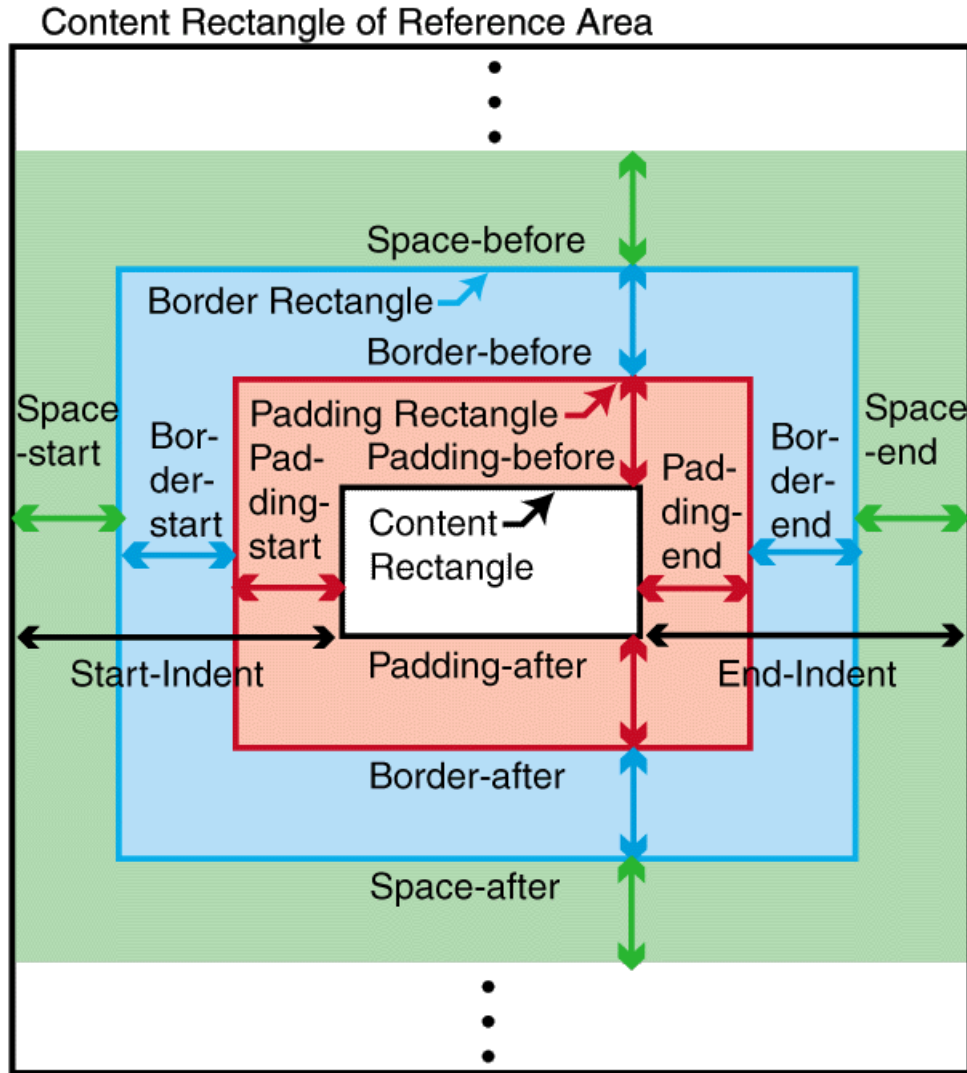
by its *start-indent* and *end-indent* and by the size of its nearest ancestor reference-area. A block-area which is not a line-area typically varies its block-progression-dimension to accommodate its descendants. Alternatively the generating formatting object may specify a block-progression-dimension for the block-area.

4.4.1. Stacked Block-areas

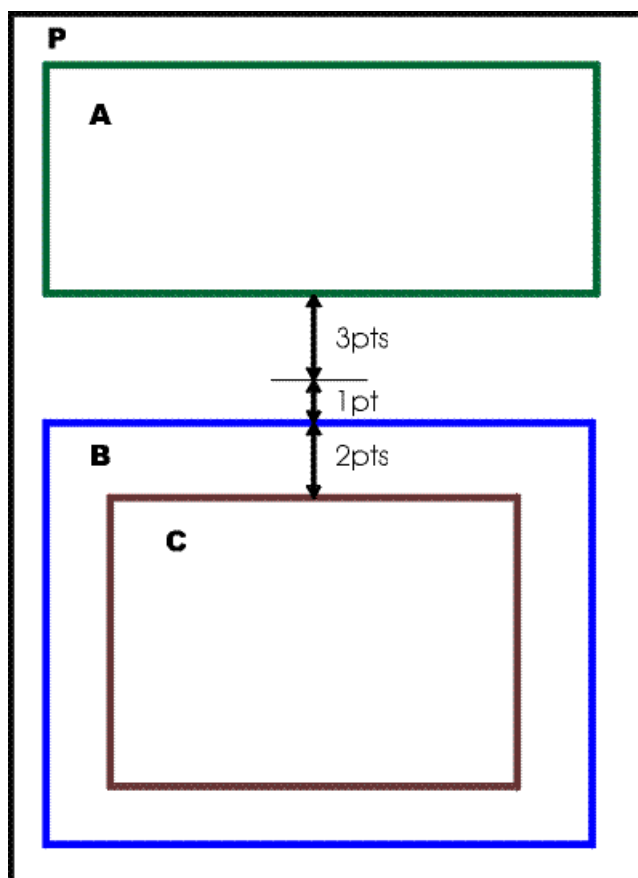
Block-area children of an area are typically stacked in the block-progression-direction within their parent area, and this is the default method of positioning block-areas. However, formatting objects are free to specify other methods of positioning child areas of areas which they generate, for example list-items or tables.

For a parent area *P* whose children are block-areas, *P* is defined to be *properly stacked* if all of the following conditions hold:

1. For each block-area *B* which is a descendant of *P*, the following hold:
 - the before-edge and after-edge of its allocation-rectangle are parallel to the before-edge and after-edges of the content-rectangle of *P*,
 - the start-edge of its allocation-rectangle is parallel to the start-edge of the content-rectangle of *R* (where *R* is the closest ancestor reference-area of *B*), and offset from it inward by a distance equal to the block-area's start-indent plus its *start-intrusion-adjustment* (as defined below), minus its border-start, padding-start, and space-start values, and
 - the end-edge of its allocation-rectangle is parallel to the end-edge of the content-rectangle of *R*, and offset from it inward by a distance equal to the block-area's end-indent plus its *end-intrusion-adjustment* (as defined below), minus its border-end, padding-end, and space-end values.
2. For each pair of normal areas *B* and *B'* in the subtree below *P*, if *B* and *B'* have a block-stacking constraint *S*, then the distance between the adjacent edges of *B* and *B'* is consistent with the constraint imposed by the resolved values of the space-specifiers in *S*.



NOTE: The *start-intrusion-adjustment* and *end-intrusion-adjustment* are traits used to deal with intrusions from floats in the inline-progression-direction. The notion of indent is intended to apply to the content-rectangle, but the constraint is written in terms of the allocation-rectangle, because as noted earlier (§ 4.2.3 – [Geometric Definitions](#) on page 16) the edges of the content-rectangle may not correspond to like-named edges of the allocation-rectangle.



Example. In the diagram, if area *A* has a space-after value of 3 points, *B* a space-before of 1 point, and *C* a space-before of 2 points, all with precedence of *force*, and with zero border and padding, then the constraints will place *B*'s allocation-rectangle 4 points below that of *A*, and *C*'s allocation-rectangle 6 points below that of *A*. Thus the 4-point gap receives the background color from *P*, and the 2-point gap before *C* receives the background color from *B*.

4.4.2. Intrusion Adjustments

Intrusion adjustments (both start- and end-) are defined to account for the indentation that occurs as the result of side floats.

If *A* and *B* are areas which have the same nearest reference area ancestor, then *A* and *B* are defined to be *inline-overlapping* if there is some line parallel to the inline-progression-direction, which intersects both the allocation-rectangle of *A* and the allocation-rectangle of *B*. If the distance in the block-progression direction from the after-edge of the allocation-rectangle of *A* to the before-edge of the allocation-rectangle of *B* some nonnegative number *y*, then *A* and *B* are defined to *have clearance of y*.

If *B* is a block-area then its *before-intrusion-allowance* is defined to be the sum of the border-before-width and padding-before-width values of all areas which are ancestors of *B* and descendants of *B*'s nearest ancestor reference-area.

If *A* is an area of class `xsl-side-float` and *B* is a block-area, and *A* and *B* have the same nearest reference area ancestor then *A* is defined to *intrude* on *B* if:

- *A* and *B* are inline-overlapping, or
- *A* and *B* have clearance of *y*, where *y* is some value greater than zero and less than the

before-intrusion-allowance of *B* (this is to account for irregularly-drawn borders and padding); or

- *A* has `float="start"`, *B* is a descendant of an area *L* generated by an `fo:list-item-body`, *A* intrudes on some line-area or reference-area descendant *D* of the sibling area of *L*, and *D* and *B* are inline-overlapping. (This is to ensure that intrusion persists long enough so that the list-item-body does not drift to the other side of the list-item-label.)

If *A* is a block-area with `float="start"`, then the start-intrusion value of *A* is the distance from the start-edge of the content-rectangle of the parent of *A* to the end-edge of the allocation-rectangle of *A*. If *A* is a block-area with `float="end"` then the end-intrusion value of *A* is the distance from the start-edge of the allocation-rectangle of *A* to the end-edge of the content-rectangle of the parent of *A*.

If *B* is a block-area which is a reference-area or a line-area, then the start-intrusion-adjustment is defined to be the maximum of the start-intrusion values of the areas which intrude on *B*. The end-intrusion-adjustment is defined to be the maximum of the end-intrusion values of the areas which intrude on *B*.

If *B* is not a reference-area or line-area, then its start-intrusion-adjustment and end-intrusion-adjustment are defined to be zero.

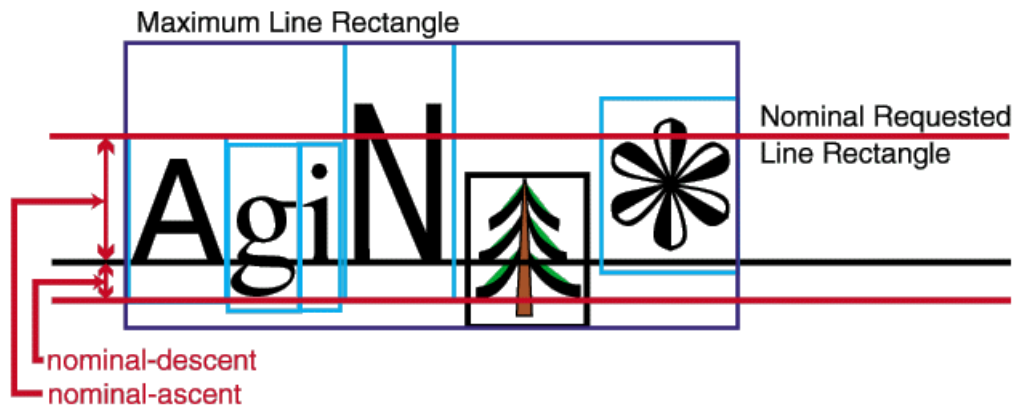
4.5. Line-areas

A line-area is a special type of block-area, and is generated by the same formatting object which generated its parent. Line-areas do not have borders and padding, i.e., `border-before-width`, `padding-before-width`, etc. are all zero. Inline-areas are stacked within a line-area relative to a *baseline-start-point* which is a point determined by the formatter, on the start-edge of the line area's content-rectangle.

The allocation-rectangle of a line is determined by the value of the *line-stacking-strategy* trait: if the value is `font-height`, the allocation-rectangle is the *nominal-requested-line-rectangle*, defined below; if the value is `max-height`, the allocation-rectangle is the *maximum-line-rectangle*, defined below; and if the value is `line-height`, the allocation-rectangle is the *per-inline-height-rectangle*, defined below.

The *nominal-requested-line-rectangle* for a line-area is the rectangle whose start-edge and end-edge are parallel to and coincident with the start-edge and end-edge of the content-rectangle of the parent block-area (as modified by typographic properties such as indents), whose before-edge is separated from the baseline-start-point by the text-altitude, and whose after-edge is separated from the baseline-start-point by the text-depth. It has the same block-progression-dimension for each line-area child of a block-area.

The *maximum-line-rectangle* for a line-area is the rectangle whose start-edge and end-edge are parallel to and coincident with the start-edge and end-edge of the nominal-requested-line-rectangle, and whose extent in the block-progression-direction is the minimum required to enclose both the nominal-requested-line-rectangle and the allocation-rectangles of all the inline-areas stacked within the line-area; this may vary depending on the descendants of the line-area.



Nominal and Maximum Line Rectangles

The *per-inline-height-rectangle* for a line-area is the rectangle whose start-edge and end-edge are parallel to and coincident with the start-edge and end-edge of the nominal-requested-line-rectangle, and whose extent in the block-progression-dimension is determined as follows. For each inline-area the *half-leading* is defined to be half the difference of its line-height minus its block-progression-dimension. The *expanded-rectangle* of an inline-area is the rectangle with start-edge and end-edge coincident with those of its allocation-rectangle, and whose before-edge and after-edge are outside those of its allocation-rectangle by a distance equal to the half-leading. The extent of the per-inline-height-rectangle in the block-progression-direction is then defined to be the minimum required to enclose both the nominal-requested-line-rectangle and the expanded-rectangles of all the inline-areas stacked within the line-area; this may vary depending on the descendants of the line-area.

NOTE: Using the nominal-requested-line-rectangle allows equal baseline-to-baseline spacing. Using the maximum-line-rectangle allows constant space between line-areas. Using the per-inline-height-rectangle and zero space-before and space-after allows CSS-style line box stacking.

4.6. Inline-areas

An inline-area has its own *line-height* trait, which may be different from the line-height of its containing block-area. This may affect the placement of its ancestor line-area when the line-stacking-strategy is *line-height*. An inline-area has an actual-baseline-table for its nominal-font. It has a *dominant-baseline-identifier* trait which determines how its stacked inline-area descendants are to be aligned.

An inline-area may or may not have child areas, and if so it may or may not be a reference-area. The dimensions of the content-rectangle for an inline-area without children is computed as specified by the generating formatting object, as are those of an inline-area with block-area children.

An inline-area with inline-area children has a content-rectangle which extends from its dominant baseline (see § 4.2.6 – [Font Baseline Tables](#) on page 25) by its after-baseline-height in the block-progression-direction, and in the opposite direction by its before-baseline-height; in the inline-progression-direction it extends from the start-edge of the allocation-rectangle of its first child to the end-end of the allocation-rectangle of its last child.

Examples of inline-areas with children might include portions of inline mathematical expressions or areas arising from mixed writing systems (left-to-right within right-to-left, for example).

4.6.1. Stacked Inline-areas

Inline-area children of an area are typically stacked in the inline-progression-direction within their parent area, and this is the default method of positioning inline-areas.

Inline-areas are stacked relative to the *dominant baseline*, as defined above (§ 4.2.6 – [Font Baseline Tables](#) on page 25).

For a parent area *P* whose children are inline-areas, *P* is defined to be *properly stacked* if all of the following conditions hold:

1. For each inline-area descendant *I* of *P*, the start-edge, end-edge, before-edge and after-edge of the allocation-rectangle of *I* are parallel to corresponding edges of the content-rectangle of the nearest ancestor reference-area of *I*.
2. For each pair of normal areas *I* and *I'* in the subtree below *P*, if *I* and *I'* have an inline-stacking constraint *S*, then the distance between the adjacent edges of *I* and *I'* is consistent with the constraint imposed by the resolved values of the space-specifiers in *S*.
3. For any inline-area descendant *I* of *P*, the distance in the shift-direction from the dominant baseline of *P* to the alignment-point of *I* equals the distance between the dominant baseline of *P* and the point corresponding to the alignment-baseline of *I* (as determined by the *actual-baseline-table* of *P*), plus the sum of the baseline-shifts for *I* and all of its ancestors which are descendants of *P*. This alignment is done with respect to the line-area's dominant baseline, and not with respect to the dominant baseline of any intermediate area.

The first summand is computed to compensate for mixed writing systems with different baseline types, and the other summands involve deliberate baseline shifts for things like superscripts and subscripts.

4.6.2. Glyph-areas

The most common inline-area is a glyph-area, which contains the representation for a character in a particular font.

A glyph-area has an associated *nominal-font*, determined by the area's typographic traits, which apply to its character data, and a *glyph-orientation* determined by its writing-mode and reference-orientation, which determine the orientation of the glyph when it is rendered.

The alignment-point and dominant-baseline-identifier of a glyph-area are assigned according to the writing-system in use (e.g., the glyph baseline in Western languages), and are used to control placement of inline-areas descendants of a line-area. The formatter may generate inline-areas with different inline-progression-directions from their parent to accommodate correct inline-area stacking in the case of mixed writing systems.

A glyph-area has no children. Its block-progression-dimension and actual-baseline-table are the same for all glyphs in a font.

4.7. Ordering Constraints

4.7.1. General Ordering Constraints

A subset *S* of the areas returned to a formatting object is called *properly ordered* if the areas in that subset have the same order as their generating formatting objects. Specifically, if *A*₁ and *A*₂ are areas in *S*, returned by child formatting objects *F*₁ and *F*₂ where *F*₁ precedes *F*₂, then *A*₁ must precede *A*₂ in the

pre-order traversal order of the area tree. If F_1 equals F_2 and A_1 is returned prior to A_2 , then A_1 must precede A_2 in the pre-order-traversal of the area tree.

For each formatting object F and each area-class C , the subset consisting of the areas returned to F with area-class C must be properly ordered, except where otherwise specified.

4.7.2. Line-building

This section describes the ordering constraints that apply to formatting an fo:block or similar block-level object.

A block-level formatting object F which constructs lines does so by constructing block-areas which it returns to its parent formatting object, and placing normal areas returned to F by its child formatting objects as children of those block-areas or of line-areas which it constructs as children of those block-areas.

For each such formatting object F , it must be possible to form an ordered partition P consisting of ordered subsets S_1, S_2, \dots, S_n of the normal areas returned by the child formatting objects, such that the following are all satisfied:

1. Each subset consists of a sequence of inline-areas, or of a single block-area.
2. The ordering of the partition follows the ordering of the formatting object tree. Specifically, if A is in S_i and B is in S_j with $i < j$, or if A and B are both in the same subset S_i with A before B in the subset order, then either A is returned by a preceding sibling formatting object of B , or A and B are returned by the same formatting object with A being returned before B .
3. The partitioning occurs at legal line-breaks. Specifically, if A is the last area of S_i and B is the first area of S_{i+1} , then the rules of the language and script in effect must permit a line-break between A and B , within the context of all areas in S_i and S_{i+1} .
4. The partition follows the ordering of the area tree, except for certain glyph substitutions and deletions. Specifically, if B_1, B_2, \dots, B_p are the normal child areas of the area or areas returned by F , (ordered in the pre-order traversal order of the area tree) then there is a one-to-one correspondence between these child areas and the partition subsets (i.e., $n = p$), and for each i ,
 - if S_i consists of a single block-area then B_i is that block-area, and
 - if S_i consists of inline-areas then B_i is a line-area whose child areas are the same as the inline-areas in S_i , and in the same order, except that where the rules of the language and script in effect call for glyph-areas to be substituted, inserted, or deleted, then the substituted or inserted glyph-areas appear in the area tree in the corresponding place, and the deleted glyph-areas do not appear in the area tree. Deletions occur when a glyph-area which is last within a subset S_i , has a *suppress-at-line-break* value of *suppress*, provided that $i < n$ and B_{i+1} is a line-area. Deletions also occur when a glyph-area which is first within a subset S_i , has a *suppress-at-line-break* value of *suppress*, provided that $i > 1$ and B_{i-1} is a line-area. Insertions and substitutions may occur because of addition of hyphens or spelling changes due to hyphenation, or glyph image construction from syllabification, or ligature formation.

Substitutions that replace a sequence of glyph-areas with a single glyph-area should only occur when the margin, border, and padding in the inline-progression-direction (start- and end-), baseline-shift, and letter-spacing values are zero, treat-as-word-space is *false*, and the values of all other relevant traits match (i.e., alignment-adjust, baseline-identifier, color trait, background traits, dominant-baseline-identifier, font traits, font-height-override-after, font-height-override-before, glyph-orientation-horizontal, glyph-orientation-vertical, line-height, line-height-shift-adjustment,

text-decoration, text-shadow).

NOTE: Line-areas do not receive the background traits or text-decoration of their generating formatting object, or any other trait that requires generation of a mark during rendering.

4.7.3. Inline-building

This section describes the ordering constraints that apply to formatting an `fo:inline` or similar inline-level object.

An inline-level formatting object *F* which constructs one or more inline-areas does so by placing normal inline-areas returned to *F* by its child formatting objects as children of inline-areas which it generates.

For each such formatting object *F*, it must be possible to form an ordered partition *P* consisting of ordered subsets *S*₁, *S*₂, ..., *S*_{*n*} of the normal inline-areas returned by the child formatting objects, such that the following are all satisfied:

1. Each subset consists of a sequence of inline-areas, or of a single block-area.
2. The ordering of the partition follows the ordering of the formatting object tree, as defined above.
3. The partitioning occurs at legal line-breaks, as defined above.
4. The partition follows the ordering of the area tree, except for certain glyph substitutions and deletions, as defined above.

4.8. Keeps and Breaks

Keep and break conditions apply to a class of areas, which are typically page-reference-areas, column-areas, and line-areas. The appropriate class for a given condition is referred to as a *context* and an area in this class is a *context-area*. As defined in Section § 6.4.1 – Introduction on page 72, *page-reference-areas* are areas generated by an `fo:page-sequence` using the specifications in a `fo:page-master`, and *column-areas* are normal-flow-reference-areas generated from a *region-body*, or *region-reference-areas* generated from other types of *region-master*.

A keep or break condition is an open statement about a formatting object and the tree relationships of the areas it generates with the relevant context-areas. These tree relationships are defined mainly in terms of *leading* or *trailing* areas. If *A* is a descendant of *P*, then *A* is defined to be *leading* in *P* if *A* has no preceding sibling which is a normal area, nor does any of its ancestor areas up to but not including *P*. Similarly, *A* is defined to be *trailing* in *P* if *A* has no following sibling which is a normal area, nor does any of its ancestor areas up to but not including *P*. For any given formatting object, the *next* formatting object in the flow is the first formatting object following (in the pre-order traversal order) which generates and returns normal areas.

Break conditions are either break-before or break-after conditions. A break-before condition is *satisfied* if the first area generated and returned by the formatting object is leading within a context-area. A break-after condition depends on the next formatting object in the flow; it is satisfied if either there is no such next formatting object, or if the first normal area generated and returned by that formatting object is leading in a context-area.

Break conditions are imposed by the *break-before* and *break-after* properties. A refined value of `page` for these traits imposes a break condition with a context consisting of the page-reference-areas; a value of `even-page` or `odd-page` imposes a break condition with a context of even-numbered page-reference-areas or odd-numbered page-reference-areas, respectively; a value of `column` imposes

a break condition with a context of column-areas. A value of `auto` in a break-before or break-after trait imposes no break condition.

Keep conditions are either keep-with-previous, keep-with-next, or keep-together conditions. A keep-with-previous condition on an object is satisfied if the first area generated and returned by the formatting object is not leading within a context-area, or if there are no preceding areas in a post-order traversal of the area tree. A keep-with-next condition is satisfied if the last area generated and returned by the formatting object is not trailing within a context-area, or if there are no following areas in a pre-order traversal of the area tree. A keep-together condition is satisfied if all areas generated and returned by the formatting object are descendants of a single context-area.

Keep conditions are imposed by the "within-page", "within-column", and "within-line" components of the "keep-with-previous", "keep-with-next", and "keep-together" properties. The refined value of each component specifies the *strength* of the keep condition imposed, with higher numbers being stronger than lower numbers and the value `always` being stronger than all numeric values. A component with value `auto` does not impose a keep condition. A "within-page" component imposes a keep-condition with context consisting of the page-reference-areas; "within-column", with context consisting of the column-areas; and "within-line" with context consisting of the line-areas.

The area tree is constrained to satisfy all break conditions imposed. Each keep condition must also be satisfied, except when this would cause a break condition or a stronger keep condition to fail to be satisfied. If not all of a set of keep conditions of equal strength can be satisfied, then some maximal satisfiable subset of conditions of that strength must be satisfied (together with all break conditions and maximal subsets of stronger keep conditions, if any).

4.9. Rendering Model

This section makes explicit the relationship between the area tree and visually rendered output.

Areas generate three types of marks: (1) the area background, if any, (2) the marks intrinsic to the area (a glyph, image, or decoration) if any, and (3) the area border, if any.

An area tree is rendered by causing marks to appear on an output medium in accordance with the areas in the area tree. This section describes the geometric location of such marks, and how conflicts between marks are to be resolved.

4.9.1. Geometry

Each area is rendered in a particular location. Formatting object semantics describe the location of intrinsic marks relative to the object's location, i.e., the left, right, top, and bottom edges of its content-rectangle. This section describes how the area's location is determined, which determines the location of its intrinsic marks.

For each page, the page-viewport-area corresponds isometrically to the output medium.

The page-reference-area is offset from the page-viewport-area as described below in section § 4.9.2 – [Viewport Geometry](#) on page 36.

All areas in the tree with an area-class of `xsl-fixed` are positioned such that the left-, right-, top-, and bottom-edges of its content-rectangle are offset inward from the content-rectangle of its ancestor page-viewport-area by distances specified by the *left-position*, *right-position*, *top-position*, and *bottom-position* traits, respectively.

Any area in the tree which is the child of a viewport-area is rendered as described in section § 4.9.2 –

[Viewport Geometry](#) on page 36.

All other areas in the tree are positioned such that the left-, right-, top-, and bottom-edges of its content-rectangle are offset inward from the content-rectangle of its nearest ancestor reference-area by distances specified by the *left-position*, *right-position*, *top-position*, and *bottom-position* traits, respectively. These are shifted left and down by the values of the *top-offset* and *left-offset* traits, respectively, if the area has a *relative-position* of *relative*.

4.9.2. Viewport Geometry

A reference-area which is the child of a viewport-area is positioned such that the start-edge and end-edge of its content-rectangle are parallel to the start-edge and end-edge of the content-rectangle of its parent viewport-area. The start-edge of its content-rectangle is offset from the start-edge of the content-rectangle of its parent viewport-area by an *inline-scroll-amount*, and the before-edge of its content-rectangle is offset from the before-edge of the content-rectangle of its parent viewport-area by a *block-scroll-amount*.

If the block-progression-dimension of the reference-area is larger than that of the viewport-area and the *overflow* trait for the reference-area is *scroll*, then the inline-scroll-amount and block-scroll-amount are determined by a scrolling mechanism, if any, provided by the user agent. Otherwise, both are zero.

4.9.3. Visibility

The visibility of marks depends upon the location of the marks, the *visibility* of the area, and the *overflow* of any ancestor viewport-areas.

If an area has visibility *hidden* it generates no marks.

If an area has an *overflow* of *hidden*, or when the environment is non-dynamic and the *overflow* is *scroll* then the area determines a *clipping rectangle*, which is defined to be the rectangle determined by the value of the *clip* trait of the area, and for any mark generated by one of its descendant areas, portions of the mark lying outside the clipping rectangle do not appear.

4.9.4. Border, Padding, and Background

The border- and padding-rectangles are determined relative to the content-rectangle by the values of the common padding and border width traits (*border-before-width*, etc.).

For any area which is not a child of a viewport-area, the border is rendered between the border-rectangle and the padding-rectangle in accordance with the common border color and style traits. For a child of a viewport-area, the border is not rendered.

For an area which is not part of a viewport/reference pair, the background is rendered. For an area that is either a viewport-area or a reference-area in a viewport/reference pair, if the refined value of *background-attachment* is *scroll* and the block-progression-dimension of the reference-area is larger than that of the viewport-area, then the background is rendered on the reference-area and not the viewport-area, and otherwise it is rendered on the viewport-area and not the reference-area.

The background, if any, is rendered in the padding-rectangle, in accordance with the *background-image*, *background-color*, *background-repeat*, *background-position-vertical*, and *background-position-horizontal* traits.

4.9.5. Intrinsic Marks

For each class of formatting objects, the marks intrinsic to its generated areas are specified in the

formatting object description. For example, an `fo:character` object generates a glyph-area, and this is rendered by drawing a glyph within that area's content-rectangle in accordance with the area's font traits and *glyph-orientation* and *blink* traits.

In addition, other traits (for example the various score and score-color traits) specify other intrinsic marks. In the case of score traits (*underline-score*, *overline-score* and *through-score*), the score thickness and position are specified by the nominal-font in effect; where the font fails to specify these quantities, they are implementation-dependent.

4.9.6. Layering and Conflict of Marks

Marks are layered as described below, which defines a partial ordering of which marks are *beneath* which other marks.

Two marks are defined to *conflict* if they apply to the same point in the output medium. When two marks conflict, the one which is beneath the other does not affect points in the output medium where they both apply.

Marks generated by the same area are layered as follows: the area background is beneath the area's intrinsic marks, and the intrinsic marks are beneath the border. Layering among the area's intrinsic marks is defined by the semantics of the area's generating formatting object and its properties. For example, a glyph-area's glyph drawing comes beneath the marks generated for text-decoration.

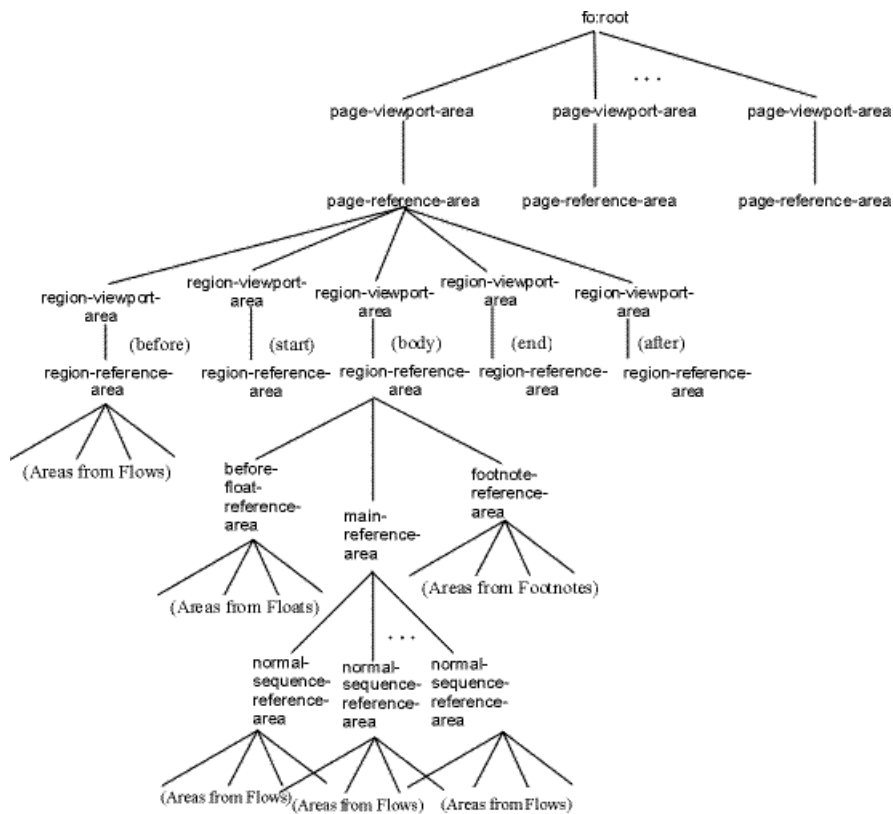
The stacking layer of an area is defined by its stacking context and its z-index value. The stacking layer of an area *A* is defined to be less than that of an area *B* if some ancestor-or-self *A'* of *A* and *B'* of *B* have the same stacking context and the z-index of *A'* is less than the z-index of *B'*. If neither stacking layer is less than the other then they are defined to have the same stacking layer.

If *A* and *B* are areas, and the stacking layer of *A* is less than the stacking layer of *B*, then all marks generated by *A* are beneath all marks generated by *B*.

If *A* and *B* are areas with the same stacking layer, the backgrounds of *A* and *B* come beneath all other marks generated by *A* and *B*. Further, if *A* is an ancestor of *B* (still with the same stacking layer), then the background of *A* is beneath all the areas of *B*, and all the areas of *B* are beneath the intrinsic areas (and border) of *A*.

If *A* and *B* have the same stacking layer and neither is an ancestor of the other, then it is an error if either their backgrounds conflict or if a non-background mark of *A* conflicts with a non-background mark of *B*. An implementation may recover by proceeding as if the marks from the first area in the pre-order traversal order are beneath those of the other area.

4.10. Sample Area Tree



A Typical Area Tree

4.11. List of Traits on Areas

- block-progression-direction
- inline-progression-direction
- shift-direction
- glyph-orientation
- is-reference-area
- is-viewport-area
- left-position
- right-position
- top-position
- bottom-position
- left-offset
- top-offset
- is-first
- is-last

- alignment-point
- area-class
- start-intrusion-adjustment
- end-intrusion-adjustment
- generated-by
- returned-by
- page-number
- blink
- underline-score
- overline-score
- through-score
- underline-score-color
- overline-score-color
- through-score-color
- alignment-baseline
- baseline-shift
- nominal-font
- dominant-baseline-identifier
- actual-baseline-table
- script

5. Property Refinement / Resolution

During refinement the set of *properties* that apply to a formatting object is transformed into a set of *traits* that define constraints on the result of formatting. For many traits there is a one-to-one correspondence with a property; for other traits the transformation is more complex. Details on the transformation are described below.

The first step in refinement of a particular formatting object is to obtain the effective value of each property that applies to the object. Any shorthand property specified on the formatting object is expanded into the individual properties. This is further described in [§ 5.2 – Shorthand Expansion](#) on page 41. For any property that has not been specified on the object the inherited (see [§ 5.1.4 – Inheritance](#) on page 41) or initial value, as applicable, is used as the effective value. The second step is to transform this property set into traits.

NOTE: Although the refinement process is described in a series of steps, this is solely for the convenience of exposition and does not imply they must be implemented as separate steps in any conforming implementation. A conforming implementation must only achieve the same effect.

5.1. Specified, Computed, and Actual Values, and Inheritance

For every property that is applicable to a given formatting object, it is necessary to determine the value of the property. Three variants of the property value are distinguished: the specified value, the computed value, and the actual value. The "specified value" is one that is placed on the formatting object during the tree-construction process. A specified value may not be in a form that is directly usable; for example, it may be a percentage or other expression that must be converted into an absolute value. A value resulting from such a conversion is called the "computed value". Finally, the computed value may not be realizable on the output medium and may need to be adjusted prior to use in rendering. For example, a line width may be adjusted to become an integral number of output medium pixels. This adjusted value is the "actual value."

5.1.1. Specified Values

The specified value of a property is determined using the following mechanisms (in order of precedence):

1. If the tree-construction process placed the property on the formatting object, use the value of that property as the specified value. This is called "explicit specification".
2. Otherwise, if the property is inheritable, use the value of that property from the parent formatting object, generally the computed value (see below).
3. Otherwise use the property's initial value, if it has one. The initial value of each property is indicated in the property's definition. If there is no initial value, that property is not specified on the formatting object. In general, this is an error.

Since it has no parent, the root of the result tree cannot use values from its parent formatting object; in this case, the initial value is used if necessary.

5.1.2. Computed Values

Specified values may be absolute (i.e., they are not specified relative to another value, as in "red" or "2mm") or relative (i.e., they are specified relative to another value, as in "auto", "2em", and "12%"), or they may be expressions. For most absolute values, no computation is needed to find the computed value. Relative values, on the other hand, must be transformed into computed values: percentages must be multiplied by a reference value (each property defines which value that is), values with a relative unit (em) must be made absolute by multiplying with the appropriate font size, "auto" values must be computed by the formulas given with each property, certain property values ("smaller", "bolder") must be replaced according to their definitions. The computed value of any property that controls a border width where the style of the border is "none" is forced to be "0pt".

Some properties have more than one way in which the property value can be specified. The simplest example of such properties are those which can be specified either in terms of a direction relative to the writing-mode (e.g., padding-before) or a direction in terms of the absolute geometric orientation of the viewport (e.g., padding-top). These two properties are called the relative property and the absolute property, respectively. Collectively, they are called "corresponding properties".

Specifying a value for one property determines both a computed value for the specified property and a computed value for the corresponding property. Which relative property corresponds to which absolute property depends on the writing-mode. For example, if the "writing-mode" at the top level of a document is "lr-tb", then "padding-start" corresponds to "padding-left", but if the "writing-mode" is "rl-tb", then "padding-start" corresponds to "padding-right". The exact specification of how to compute

the values of corresponding properties is given in [§ 5.3 – Computing the Values of Corresponding Properties](#) on page 42.

In most cases, elements inherit computed values. However, there are some properties whose specified value may be inherited (e.g., some values for the "line-height" property). In the cases where child elements do not inherit the computed value, this is described in the property definition.

5.1.3. Actual Values

A computed value is in principle ready to be used, but a user agent may not be able to make use of the value in a given environment. For example, a user agent may only be able to render borders with integer pixel widths and may, therefore, have to adjust the computed width to an integral number of media pixels. The actual value is the computed value after any such adjustments have been applied.

5.1.4. Inheritance

Some of the properties applicable to formatting objects are "inheritable." Such properties are so identified in the property description. The inheritable properties can be placed on any formatting object. The inheritable properties are propagated down the formatting object tree from a parent to each child. (These properties are given their initial value at the root of the result tree.) For a given inheritable property, if that property is present on a child, then that value of the property is used for that child (and its descendants until explicitly re-set in a lower descendant); otherwise, the specified value of that property on the child is the computed value of that property on the parent formatting object. Hence there is always a specified value defined for every inheritable property for every formatting object.

5.2. Shorthand Expansion

In XSL there are two kinds of shorthand properties; those originating from CSS, such as "border", and those that arise from breaking apart and/or combining CSS properties, such as "page-break-inside". In XSL both types of shorthands are handled in the same way.

NOTE: Shorthands are only included in the highest XSL conformance level: "complete".

The conformance level for each property is shown in [Appendix C.3 – Property Table: Part II](#) on page 365.

Shorthand properties do not inherit from the shorthand on the parent. Instead the individual properties that the shorthand expands into may inherit.

Some CSS shorthands are interrelated; their expansion has one or more individual properties in common. CSS indicates that the user must specify the order of processing for combinations of multiple interrelated shorthands and individual interrelated properties. In XML, attributes are defined as unordered. To resolve this issue, XSL defines a precedence order when multiple interrelated shorthand properties or a shorthand property and an interrelated individual property are specified:

They are processed in increasing precision (i.e., "border" is less precise than "border-top", which is less precise than "border-top-color"). The individual properties are always more precise than any shorthand. For the remaining ambiguous case, XSL defines the ordering to be:

1. "border-style", "border-color", and "border-width" is less precise than
2. "border-top", "border-bottom", "border-right", and "border-left".

Processing is conceptually in the following steps:

1. Set the effective value of all properties to their initial values.

2. Process all shorthands in increasing precision.

If the shorthand is set to "inherit": set the effective value of each property that can be set by the shorthand to the computed value of the corresponding property in the parent.

If the value of the shorthand is not "inherit": determine which individual properties are to be set, and replace the initial value with the computed value derived from the specified value.

3. Process all specified individual properties.

4. Carry out any inheritance for properties that were not given a value other than by the first step.

NOTE: For example, if both the "background" and the "background-color" properties are specified on a given formatting object: process the "background" shorthand, then process the "background-color" property.

5.3. Computing the Values of Corresponding Properties

Where there are corresponding properties, such as "padding-left" and "padding-start", a computed value is determined for all the corresponding properties. How the computed values are determined for a given formatting object is dependent on which of the corresponding properties are specified on the object. See description below.

The correspondence mapping from absolute to relative property is as follows:

If the "writing-mode" specifies a block-progression-direction of "top-to-bottom": "top" maps to "before", and "bottom" maps to "after".

If the "writing-mode" specifies a block-progression-direction of "bottom-to-top": "top" maps to "after", and "bottom" maps to "before".

If the "writing-mode" specifies a block-progression-direction of "left-to-right": "left" maps to "before", and "right" maps to "after".

If the "writing-mode" specifies a block-progression-direction of "right-to-left": "left" maps to "after", and "right" maps to "before".

If the "writing-mode" specifies an inline-progression-direction of "left-to-right": "left" maps to "start", and "right" maps to "end".

If the "writing-mode" specifies an inline-progression-direction of "right-to-left": "left" maps to "end", and "right" maps to "start".

If the "writing-mode" specifies an inline-progression-direction of "top-to-bottom": "top" maps to "start", and "bottom" maps to "end".

If the "writing-mode" specifies an inline-progression-direction of "bottom-to-top": "top" maps to "end", and "bottom" maps to "start".

If the "writing-mode" specifies an inline-progression-direction of "left-to-right" for odd-numbered lines, and "right-to-left" for even-numbered lines: "left" maps to "start", and "right" maps to "end".

NOTE: "reference-orientation" is a rotation and does not influence the correspondence mapping.

5.3.1. Border and Padding Properties

The simplest class of corresponding properties are those for which there are only two variants in the

correspondence, an absolute property and a relative property, and the property names differ only in the choice of absolute or relative designation; for example, "border-left-color" and "border-start-color".

For this class, the computed values of the corresponding properties are determined as follows. If the corresponding absolute variant of the property is specified on the formatting object, its computed value is used to set the computed value of the corresponding relative property. If the corresponding absolute property is not explicitly specified, then the computed value of the absolute property is set to the computed value of the corresponding relative property.

Note that if both the absolute and the relative properties are not explicitly specified, then the rules for determining the specified value will use either inheritance if that is defined for the property or the initial value. The initial value must be the same for all possible corresponding properties. If both an absolute and a corresponding relative property are explicitly specified, then the above rule gives precedence to the absolute property, and the specified value of the corresponding relative property is ignored in determining the computed value of the corresponding properties.

The (corresponding) properties that use the above rule to determine their computed value are:

- border-after-color
- border-before-color
- border-end-color
- border-start-color
- border-after-style
- border-before-style
- border-end-style
- border-start-style
- border-after-width
- border-before-width
- border-end-width
- border-start-width
- padding-after
- padding-before
- padding-end
- padding-start

5.3.2. Margin, Space, and Indent Properties

The "space-before", and "space-after" properties (block-level formatting objects), "space-start", and "space-end" properties (inline-level formatting objects) are handled in the same way as the properties immediately above, but the corresponding absolute properties are in the set: "margin-top", "margin-bottom", "margin-left", and "margin-right".

There are two more properties, "end-indent" and "start-indent" (block-level formatting objects), for which the computed value may be determined by the computed value of the absolute margin properties. For these traits, the calculation of the value of the trait when the corresponding absolute property is

present depends on three computed values: the computed value of the corresponding absolute property, the computed value of the corresponding "padding" property, and the computed value of the corresponding "border-width" property.

Here the term "corresponding" has been broadened to mean that if "margin-left" is the corresponding absolute property to "start-indent", then "padding-left" (and "padding-start") and "border-left-width" (and "border-start-width") are the "corresponding" "padding" and "border-width" properties.

The formulae for calculating the computed value of the "start-indent", and "end-indent" properties are as follows (where "margin-corresponding" is a variable for the corresponding absolute "margin" property):

```
end-indent = margin-corresponding + padding-end + border-end-width
start-indent = margin-corresponding + padding-start + border-start-width
```

If an absolute "margin" property is not explicitly specified, these equations determine a computed value for the corresponding "margin" property given values for the three traits *corresponding-indent*, *padding-corresponding* and *border-corresponding* width.

5.3.3. Height, and Width Properties

Based on the writing-mode in effect for the formatting object, either the "height", "min-height", and "max-height" properties, or the "width", "min-width", and "max-width" properties are converted to the corresponding block-progression-dimension, or inline-progression-dimension.

The "height" properties are absolute and indicate the dimension from "top" to "bottom"; the width properties the dimension from "left" to "right".

If the "writing-mode" specifies a block-progression-direction of "top-to-bottom" or "bottom-to-top" the conversion is as follows:

- If any of "height", "min-height", or "max-height" is specified:
 - If "height" is specified then first set:
 - block-progression-dimension.minimum=<height>
 - block-progression-dimension.optimum=<height>
 - block-progression-dimension.maximum=<height>
 - If "height" is not specified, then first set:
 - block-progression-dimension.minimum=auto
 - block-progression-dimension.optimum=auto
 - block-progression-dimension.maximum=auto
 - Then, if "min-height" is specified, reset:
 - block-progression-dimension.minimum=<min-height>
 - Then, if "max-height" is specified, reset:
 - block-progression-dimension.minimum=<max-height>
 - However, if "max-height" is specified as "none", reset:
 - block-progression-dimension.minimum=auto

- If any of "width", "min-width", or "max-width" is specified:
 - If "width" is specified then first set:
inline-progression-dimension.minimum=<width>
inline-progression-dimension.optimum=<width>
inline-progression-dimension.maximum=<width>
 - If "width" is not specified, then first set:
inline-progression-dimension.minimum=auto
inline-progression-dimension.optimum=auto
inline-progression-dimension.maximum=auto
 - Then, if "min-width" is specified, reset:
inline-progression-dimension.minimum=<min-width>
 - Then, if "max-width" is specified, reset:
inline-progression-dimension.minimum=<max-width>
 - However, if "max-width" is specified as "none", reset:
inline-progression-dimension.minimum=auto

If the "writing-mode" specifies a block-progression-direction of "left-to-right" or "right-to-left" the conversion is as follows:

- If any of "height", "min-height", or "max-height" is specified:
 - If "height" is specified then first set:
inline-progression-dimension.minimum=<height>
inline-progression-dimension.optimum=<height>
inline-progression-dimension.maximum=<height>
 - If "height" is not specified, then first set:
inline-progression-dimension.minimum=auto
inline-progression-dimension.optimum=auto
inline-progression-dimension.maximum=auto
 - Then, if "min-height" is specified, reset:
inline-progression-dimension.minimum=<min-height>
 - Then, if "max-height" is specified, reset:
inline-progression-dimension.minimum=<max-height>
 - However, if "max-height" is specified as "none", reset:
inline-progression-dimension.minimum=auto

- If any of "width", "min-width", or "max-width" is specified:
 - If "width" is specified then first set:
 - block-progression-dimension.minimum=<width>
 - block-progression-dimension.optimum=<width>
 - block-progression-dimension.maximum=<width>
 - If "width" is not specified, then first set:
 - block-progression-dimension.minimum=auto
 - block-progression-dimension.optimum=auto
 - block-progression-dimension.maximum=auto
 - Then, if "min-width" is specified, reset:
 - block-progression-dimension.minimum=<min-width>
 - Then, if "max-width" is specified, reset:
 - block-progression-dimension.maximum=<max-width>
 - However, if "max-width" is specified as "none", reset:
 - block-progression-dimension.maximum=auto

5.3.4. Overconstrained Geometry

The sum of the start-indent, end-indent, and inline-progression-dimension of the content-rectangle of an area should be equal to the inline-progression-dimension of the content-rectangle of the closest ancestor reference-area. In the case where a specification would lead to them being different the end-indent (and thus the corresponding margin) is adjusted such that the equality is true.

5.4. Simple Property to Trait Mapping

The majority of the properties map into traits of the same name. Most of these also simply copy the value from the property. These are classified as "Rendering", "Formatting", "Specification", "Font selection", "Reference", and "Action" in the property table in [Appendix C.3 – Property Table: Part II](#) on page 365. For example, the property `font-style="italic"` is refined into a *font-style* trait with a value of "italic".

Some traits have a value that is different from the value of the property. These are classified as "Value change" in the property table. For example, the property `background-position-horizontal="left"` is refined into a *background-position-horizontal* trait with a value of "Opt". The value mapping for these traits is given below.

5.4.1. Background-position-horizontal and background-position-vertical Properties

A value of "top", "bottom", "left", "right", or "center" is converted to a length as specified in the property definition.

5.4.2. Column-number Property

If a value has not been specified on a formatting object to which this property applies the initial value is

computed as specified in the property definition.

5.4.3. Text-align Property

A value of "left", or "right" is converted to the writing-mode relative value as specified in the property definition.

5.4.4. Text-align-last Property

A value of "left", or "right" is converted to the writing-mode relative value as specified in the property definition.

5.4.5. z-index Property

The value is converted to one that is absolute; i.e., the refined value is the specified value plus the refined value of z-index of its parent formatting object, if any.

5.5. Complex Property to Trait Mapping

A small number of properties influence traits in a more complex manner. Details are given below.

5.5.1. Word-spacing and Letter-spacing Properties

These properties may set values for the *start-space* and *end-space* traits, as described in the property definitions.

5.5.2. Reference-orientation Property

- If this area sets a reference-area:
 1. Let *A* be the area whose traits are being determined;
 2. Let *F* be the formatting object that generates area *A*;
 3. Let *N* be the nearest ancestor formatting object, including self, to *F* such that *N* generates a reference-area that is an ancestor of *A* in the area tree:

Then, the *reference-orientation* trait for *A* is set to the modulo-360 sum of the "reference-orientation" property on *F* and the value established by the "reference-orientation" property of formatting object *N* (recursively).

- All other areas:
 1. Let *A* be the area whose traits are being determined;
 2. Let *F* be the formatting object that generates area *A*;
 3. Let *N* be the nearest ancestor formatting object, including self, to *F* such that *N* generates a reference-area that is an ancestor of *A* in the area tree:

Then, the *reference-orientation* trait for *A* is set to the value established by the "reference-orientation" property of formatting object *N*.

5.5.3. Writing-mode and Direction Properties

The direction traits on an area are indirectly derived from the "writing-mode", "direction" and "unicode-bidi" properties on the formatting object that generates the area or the formatting object

ancestors of that formatting object. The exact derivation depends on the trait.

- block-progression-direction trait:
 - Let *A* be the area whose traits are being determined;
 - Let *F* be the formatting object that generates area *A*;
 - Let *N* be the nearest ancestor formatting object, including self, to *F* such that *N* generates a reference-area that is an ancestor of *A* in the area tree:

Then, the *block-progression-direction* trait for *A* is set to the block-progression-direction established by the "writing-mode" property of formatting object *N*.

- inline-progression-direction trait:
 - Let *A* be the area whose traits are being determined;
 - Let *F* be the formatting object that generates area *A*;
 - Let *N* be the nearest ancestor formatting object, including self, to *F* such that at least one of the following conditions is satisfied:
 - *N* generates a reference-area that is an ancestor to *A* in the area tree, or
 - *N* is a block-level formatting object and the glyph-orientation of *N* is either 0 or 180 degrees, in which case, the "direction" property establishes the inline-progression-direction for *N*, or

NOTE: The above condition is complex because direction changes in vertical text only affect the rotated non-ideographic text; that is they only affect the Latin or Arabic/Hebrew half-width characters.

- *N* is an inline formatting object and has a "unicode-bidi" property with a value of either "embed" or "override", in which case, the "direction" property establishes an inline-progression-direction for *N*.

Then, the *inline-progression-direction* trait for *A* is set to the inline-progression-direction established by formatting object *N*. If *N* has both a "writing-mode" and a "direction" property that establishes the inline-progression-direction then the "direction" property's overrides that established by the "writing-mode".

The final *inline-progression-direction-trait* may be further modified by the formatter's application of the unicode-bidi rule, and for alternating lines in the extended set of international writing-modes.

- shift-direction trait:
 - Let *A* be the area whose traits are being determined;
 - Let *F* be the formatting object that generates area *A*;
 - Let *N* be the nearest ancestor formatting object, including self, to *F* such that *N* generates a reference-area that is an ancestor of *A* in the area tree:

Then, the *shift-direction* trait for *A* is set to the shift-direction established by the "writing-mode" property of formatting object *N*.

The *shift-direction-trait* may be further modified by the formatter for inverted lines in the extended set of international writing-modes.

5.5.4. Absolute-position Property

If `absolute-position` = "absolute" or "fixed", the values of the *left-position*, *top-position*, etc. traits are copied directly from the values of the "left", "top", etc. properties. Otherwise these traits' values are left undefined during refinement and determined during composition.

5.5.5. Relative-position Property

If `relative-position` = "relative" then the values of the *left-offset* and *top-offset* traits are copied directly from the "left" and "top" properties. If the "right" property is specified but "left" is not, then *left-offset* is set to the negative of the value of "right". If neither "left" nor "right" is specified the *left-offset* is 0. If the "bottom" property is specified but "top" is not, then *top-offset* is set to the negative of the value of "bottom". If neither "top" nor "bottom" is specified the *top-offset* is 0.

5.5.6. Text-decoration Property

The "text-decoration" property value provides values for the *blink* trait and a set of score and score-color traits. The *specified color* has the value of the *color* trait of the formatting object for which the "text-decoration" property is being refined.

A property value containing the token "underline" sets a value of "yes" to the *underline-score* trait, and a value of *specified color* to the *underline-score-color* trait.

A property value containing the token "overline" sets a value of "yes" to the *overline-score* trait, and a value of *specified color* to the *overline-score-color* trait.

A property value containing the token "line-through" sets a value of "yes" to the *through-score* trait, and a value of *specified color* to the *through-score-color* trait.

A property value containing the token "blink" sets a value of "yes" to the *blink* trait.

A property value containing the token "no-underline" sets a value of "no" to the *underline-score* trait, and a value of *specified color* to the *underline-score-color* trait.

A property value containing the token "no-overline" sets a value of "no" to the *overline-score* trait, and a value of *specified color* to the *overline-score-color* trait.

A property value containing the token "no-line-through" sets a value of "no" to the *through-score* trait, and a value of *specified color* to the *through-score-color* trait.

A property value containing the token "no-blink" sets a value of "no" to the *blink* trait.

5.5.7. Font Properties

The font traits on an area are indirectly derived from the combination of the font properties, which are used to select a font, and the font tables from that font.

The abstract model that XSL assumes for a font is described in § 7.7.1 – [Fonts and Font Data](#) on page 190.

There is no XSL mechanism to specify a particular font; instead, a *selected font* is chosen from the fonts available to the User Agent based on a set of selection criteria. The *selection criteria* are the following font properties: "font-family", "font-style", "font-variant", "font-weight", "font-stretch", and "font-size", plus, for some formatting objects, one or more characters. The details of how the selection criteria are used is specified in the "font-selection-strategy" property (see § 7.7.3 – [“font-selection-strategy”](#) on page 194).

The *nominal-font* trait is set to the selected font. In the case where there is no selected font and the 'missing character' glyph is displayed, the *nominal-font* trait is set to the font containing that glyph, otherwise (i.e., some other mechanism was used to indicate that a character is not being displayed) the *nominal-font* is a system font.

The *dominant-baseline-identifier* and *actual-baseline-table* traits are derived from the value of the "dominant-baseline" property. The value of this property is a compound value with three components: a baseline-identifier for the dominant-baseline, a baseline-table and a baseline-table font-size. The *dominant-baseline-identifier* is set from the first component. The baseline-table font-size is used to scale the the positions of the baselines from the baseline table and, then, the position of the dominant-baseline is subtracted from the positions of the other baselines to yield a table of offsets from the dominant baseline. This table is the value of the *actual-baseline-table* trait.

5.6. Non-property Based Trait Generation

The *is-reference-area* trait is set to "true" for the following formatting objects: "simple-page-master", "title", "region-body", "region-before", "region-after", "region-start", "region-end", "block-container", "inline-container", "table", "table-caption", and "table-cell". For all other formatting objects it is set to "false".

5.7. Property Based Transformations

5.7.1. Text-transform Property

The case changes specified by this property are carried out during refinement by changing the value of the "character" property appropriately.

NOTE: The use of the "text-transform" property is deprecated in XSL due to its severe internationalization issues.

5.8. Unicode Bidi Processing

NOTE: This section is subject to further updating.

The final step of refinement uses the Unicode Bidi Algorithm [UNICODE TR9] and the Unicode bidirectional character type of each character to convert the implicit directionality of the text into explicit markup in terms of formatting objects. For example, sequences of Arabic text are wrapped in formatting objects with a "direction" property of "rtl" and a "unicode-bidi" property of "bidi-override".

As defined in the referenced specification, the Unicode Bidi Algorithm takes a stream of text as input, and proceeds in three main phases:

1. Separation of the input text into paragraphs. The rest of the algorithm affects only the text between paragraph separators.
2. Resolution of the embedding levels of the text. In this phase, the bidirectional character types, plus the Unicode directional formatting codes, are used to produce *resolved embedding levels*. The normative bidirectional character type for each character is specified in the Unicode Character Database [UNICODE Character Database].
3. Reordering the text for display on a line-by-line basis using the resolved embedding levels, once the text has been broken into lines.

Specifying how this algorithm applies to XSL requires some elaboration. First, the final, text reordering

step is not done; instead, the placement of glyphs as defined in the Area Model uses the *inline-progression-direction* of each glyph to control the stacking of glyphs as described in § 4.2.5 – [Stacking Constraints](#) on page 19. The *inline-progression-direction* is derived from the "direction" property of the formatting objects inserted in this step of refinement.

Second, the algorithm is applied to a sequence of characters coming from the content of one or more formatting objects. The sequence of characters is created by processing a fragment of the formatting object tree. A *fragment* is any contiguous sequence of children of some formatting object in the tree. The sequence is created by doing a pre-order traversal of the fragment down to the fo:character level. During the pre-order traversal, every fo:character formatting object adds a character to the sequence. Furthermore, whenever the pre-order scan encounters a node with a "unicode-bidi" property with a value of "embed" or "override", add a Unicode RLO/LRO or RLE/LRE character to the sequence as appropriate to the value of the "direction" and "unicode-bidi" properties. On returning to that node after traversing its content, add a Unicode PDF character. In this way, the formatting object tree fragment is flattened into a sequence of characters. This sequence of characters is called the *flattened sequence of characters* below.

Third, in XSL the algorithm is applied to *delimited text ranges* instead of just paragraphs. A delimited text range is a maximal flattened sequence of characters that does not contain any delimiters. Any formatting object that generates block areas is a delimiter. It acts as a delimiter for its content. It also acts as a delimiter for its parent's content. That is, if the parent has character content, then its children formatting objects that generate block areas act to break that character content into *anonymous blocks* each of which is a delimited text range. In a similar manner, the fo:multi-case formatting object acts as delimiter for its content and the content of its parent. Finally, text with an orientation that is not perpendicular to the dominant-baseline acts as a delimiter to text with an orientation perpendicular to the dominant-baseline. We say that *text has an orientation perpendicular to the dominant-baseline* if the glyphs that correspond to the characters in the text are all oriented perpendicular to the dominant-baseline.

NOTE: In most cases, a delimited text range is the maximal sequence of characters that would be formatted into a sequence of one or more line-areas. For the fo:multi-case and the text with an orientation perpendicular to the dominant-baseline, the delimited range may be a sub-sequence of a line or sequence of lines. For example, in Japanese formatted in a vertical writing-mode, rotated Latin and Arabic text would be delimited by the vertical Japanese characters that immediately surround the Latin and Arabic text. Any formatting objects that generated inline-areas would have no affect on the determination of the delimited text range.

For each delimited text range, the *inline-progression-direction* of the nearest ancestor (including self) formatting object that generates a block-area determines the *paragraph embedding level* used in the Unicode Bidi Algorithm. This is the default embedding level for the delimited text range.

Embedding levels are numbers that indicate how deeply the text is nested, and the default direction of text on that level. The minimum embedding level of text is zero, and the maximum explicit depth is level 61. The second step of the Unicode Bidi Algorithm labels each character in the delimited text range with a *resolved embedding level*. The resolved embedding level of each character will be greater than or equal to the paragraph embedding level. Right-to-left text will always end up with an odd level, and left-to-right and numeric text will always end up with an even level. In addition, numeric text will always end up with a higher level than the paragraph level.

Once the resolved embedding levels are determined for the delimited text range, new fo:inline formatting objects with appropriate values for the "direction" and "unicode-bidi" properties are inserted into the formatting object tree fragment that was flattened into the delimited text range such that the following constraints are satisfied:

1. For any character in the delimited text range, the *inline-progression-direction* of the character must match its resolved embedding level.
2. For each resolved embedding level L from the paragraph embedding level to the maximum resolved embedding level, and for each maximal contiguous sequence of characters S for which the resolved embedding level of each character is greater than or equal to L ,
 - A. There is an inline formatting object F which has as its content the formatting object tree fragment that flattens to S and has a "direction" property consistent with the resolved embedding level L .

NOTE: F need not be an inserted formatting object if the constraint is met by an existing formatting object or by specifying values for the "direction" and "unicode-bidi" properties on an existing formatting object.

- B. All formatting objects that contain any part of the sequence S are properly nested in F and retain the nesting relationships they had in the formatting object tree prior to the insertion of the new formatting objects.

NOTE: Satisfying this constraint may require splitting one or more existing formatting objects in the formatting object tree each into a pair of formatting objects each of which has the same set of computed property values as the original, unsplit formatting object. One of the pair would be ended before the start of F or start after the end of F and the other would start after the start of F or would end before the end of F , respectively. The created pairs must continue to nest properly to satisfy this constraint. For example, assume Left-to-right text is represented by the character "L" and Right-to-left text is represented by "R". In the sub-tree

```
<fo:block>
  LL
  <fo:inline ID="A">LLLRRL</fo:inline>
  RR
</fo:block>
```

assuming a paragraph embedding level of "0", the resolved embedding levels would require the following (inserted and replicated) structure:

```
<fo:block>
  LL
  <fo:inline ID="A">LLL</fo:inline>
  <fo:inline direction="rtl">
    <fo:inline ID="A+">RRL</fo:inline>
  RR
</fo:inline>
</fo:block>
```

Note that the fo:inline with ID equal "A" has been split into two fo:inlines with the first one having the original ID of "A" and the second having an ID of "A+". Since ID's must be unique, the computed value of any ID or Key property must not be replicated in the second member of the pair. The value of "A+" was just used for illustrative purposes.

3. No fewer fo:inline formatting objects can be inserted and still satisfy the above constraints.

5.9. Expressions

All property value specifications in attributes within an XSL stylesheet can be expressions. These expressions represent the value of the property specified. The expression is first evaluated and then the resultant value is used to determine the value of the property.

5.9.1. Property Context

Properties are evaluated against a property-specific context. This context provides:

- A list of allowed resultant types for a property value.
- Conversions from resultant expression value types to an allowed type for the property.
- The current font-size value.
- Conversions from relative numerics by type to absolute numerics within additive expressions.

NOTE: It is not necessary that a conversion is provided for all types. If no conversion is specified, it is an error.

When a type instance (e.g., a string, a keyword, a numeric, etc.) is recognized in the expression it is evaluated against the property context. This provides the ability for specific values to be converted with the property context's specific algorithms or conversions for use in the evaluation of the expression as a whole.

For example, the "auto" enumeration token for certain properties is a calculated value. Such a token would be converted into a specific type instance via an algorithm specified in the property definition. In such a case the resulting value might be an absolute length specifying the width of some aspect of the formatting object.

In addition, this allows certain types like relative numerics to be resolved into absolute numerics prior to mathematical operations.

All property contexts allow conversions as specified in § 5.9.12 – Expression Value Conversions on page 57.

5.9.2. Evaluation Order

When a set of properties is being evaluated for a specific formatting object in the formatting object tree there is a specific order in which properties must be evaluated. Essentially, the "font-size" property must be evaluated first before all other properties. Once the "font-size" property has been evaluated, all other properties may be evaluated in any order.

When the "font-size" property is evaluated, the current font-size for use in evaluation is the font-size of the parent element. Once the "font-size" property has been evaluated, that value is used as the current font-size for all property contexts of all properties value expressions being further evaluated.

5.9.3. Basics

- [1] Expr ::= AdditiveExpr
- [2] PrimaryExpr ::= '(' Expr ')'
 | Numeric
 | Literal
 | Color
 | Keyword
 | EnumerationToken
 | FunctionCall

5.9.4. Function Calls

[3] FunctionCall ::= **FunctionName** '(' (**Argument** (',' **Argument**)*)? ')'

[4] Argument $::=$ Expr

5.9.5. Numerics

A numeric represents all the types of numbers in an XSL expression. Some of these numbers are absolute values. Others are relative to some other set of values. All of these values use a floating-point number to represent the number-part of their definition.

A floating-point number can have any double-precision 64-bit format IEEE 754 value [IEEE 754]. These include a special “Not-a-Number” (NaN) value, positive and negative infinity, and positive and negative zero. See Section 4.2.3 of [JLS] for a summary of the key rules of the IEEE 754 standard.

```
[5] Numeric ::= AbsoluteNumeric
          | RelativeNumeric
```

[6] AbsoluteNumeric ::= **AbsoluteLength**

[7] `AbsoluteLength` ::= **Number** `AbsoluteUnitName`?

[8] RelativeNumeric ::= Percent
 | RelativeLength

[9] Percent ::= **Number** '%'

[10] RelativeLength ::= **Number** RelativeUnitName

The following operators may be used with numerics:

+ Performs addition.

- Performs subtraction or negation.

- * Performs multiplication.

div Performs floating-point division according to IEEE 754.

mod Returns the remainder from a truncating division.

NOTE: Since XML allows – in names, the – operator (when not used as a UnaryExpr negation) typically needs to be preceded by whitespace. For example the expression 10pt – 2pt means subtract 2 points from 10 points. The expression 10pt–2pt would mean a length value of 10 with a unit of "pt-2pt".

NOTE: The following are examples of the mod operator:

- $5 \bmod 2$ returns 1
- $5 \bmod -2$ returns 1
- $-5 \bmod 2$ returns -1
- $-5 \bmod -2$ returns -1

NOTE: The mod operator is the same as the % operator in Java and ECMAScript and is not the same as the IEEE remainder operation, which returns the remainder from a rounding division.

Numeric Expressions

```

[11]      AdditiveExpr ::= MultiplicativeExpr
                        | AdditiveExpr '+' MultiplicativeExpr
                        | AdditiveExpr '-' MultiplicativeExpr

[12]      MultiplicativeExpr ::= UnaryExpr
                                | MultiplicativeExpr MultiplyOperator UnaryExpr
                                | MultiplicativeExpr 'div' UnaryExpr
                                | MultiplicativeExpr 'mod' UnaryExpr

[13]      UnaryExpr ::= PrimaryExpr
                    | '-' UnaryExpr

```

NOTE: The effect of this grammar is that the order of precedence is (lowest precedence first):

- $+$, $-$
- $*$, div , mod

and the operators are all left associative. For example, $2*3 + 4 \text{ div } 5$ is equivalent to $(2*3) + (4 \text{ div } 5)$.

If a non-numeric value is used in an **AdditiveExpr** and there is no property context conversion from that type into an absolute numeric value, the expression is invalid and considered an error.

5.9.6. Absolute Numerics

An absolute numeric is an absolute length which is a pair consisting of a **Number** and a **UnitName** raised to a power. When an absolute length is written without a unit, the unit power is assumed to be zero. Hence, all floating point numbers are a length with a power of zero.

Each unit name has associated with it an internal ratio to some common internal unit of measure (e.g., a meter). When a value is written in a property expression, it is first converted to the internal unit of measure and then mathematical operations are performed.

In addition, only the mod, addition, and subtraction operators require that the numerics on either side of the operation be absolute numerics of the same unit power. For other operations, the unit powers may be different and the result should be mathematically consistent as with the handling of powers in algebra.

A property definition may constrain an absolute length to a particular power. For example, when specifying font-size, the value is expected to be of power "one". That is, it is expected to have a single powered unit specified (e.g., 10pt).

When the final value of property is calculated, the resulting power of the absolute numeric must be either zero or one. If any other power is specified, the value is an error.

5.9.7. Relative Numerics

Relative lengths are values that are calculated relative to some other set of values. When written as part of an expression, they are either converted via the property context into an absolute numeric or passed verbatim as the property value.

It is an error if the property context has no available conversion for the relative numeric and a conversion is required for expression evaluation (e.g., within an add operation).

5.9.7.1. Percents

Percentages are values that are counted in 1/100 units. That is, 10% as a percentage value is 0.10 as a floating point number. When converting to an absolute numeric, the percentage is defined in the property definition as being a percentage of some known property value.

For example, a value of "110%" on a "font-size" property would be evaluated to mean 1.1 times the current font size. Such a definition of the allowed conversion for percentages is specified on the property definition. If no conversion is specified, the resulting value is a percentage.

5.9.7.2. Relative Lengths

A relative length is a unit-based value that is measured against the current value of the `font-size` property.

There is only one relative unit of measure, the "em". The definition of "1em" is equal to the current font size. For example, a value of "1.25em" is 1.25 times the current font size.

When an em measurement is used in an expression, it is converted according to the `font-size` value of the current property's context. The result of the expression is an absolute length. See § 7.7.4 – “font-size” on page 195

5.9.8. Strings

Strings are represented either as **literals** or as an **enumeration token**. All properties contexts allow conversion from enumeration tokens to strings. See § 5.9.12 – Expression Value Conversions on page 57.

5.9.9. Colors

A color is a set of values used to identify a particular color from a color space. Only RGB (Red, Green, Blue) and ICC (International Color Consortium) colors are included in this Recommendation.

RGB colors are directly represented in the expression language using a hexadecimal notation. ICC colors can be accessed through an `icc-color` function. Colors can also be accessed through the `system-color` function or through conversion from an **EnumerationToken** via the property context.

5.9.10. Keywords

Keywords are special tokens in the grammar that provide access to calculated values or other property values. The allowed keywords are defined in the following subsections.

5.9.10.1. inherit

The property takes the same *computed* value as the property for the formatting object's parent object.

5.9.11. Lexical Structure

When processing an expression, whitespace (**ExprWhitespace**) may be allowed before or after any expression token even though it is not explicitly defined as such in the grammar. In some cases, whitespace is necessary to make tokens in the grammar lexically distinct. Essentially, whitespace should be treated as if it does not exist after tokenization of the expression has occurred.

The following special tokenization rules must be applied in the order specified to disambiguate the grammar:

- If the character following an **NCName** (possibly after intervening **ExprWhitespace**) is " (", then the token must be recognized as **FunctionName**.
- A number terminates at the first occurrence of a non-digit character other than ".". This allows the unit token for length quantities to parse properly.
- When an **NCName** immediately follows a **Number**, it should be recognized as a **UnitName** or it is an error.
- The **Keyword** values take precedence over **EnumerationToken**.
- If a **NCName** follows a **numeric**, it should be recognized as an **OperatorName** or it is an error.

Expression Lexical Structure

[14]	ExprToken	::=	'(' ')' '%' Operator FunctionName EnumerationToken Number
[15]	Number	::=	FloatingPointNumber
[16]	FloatingPointNumber	::=	Digits ('.' Digits)? '.' Digits
[17]	Digits	::=	[0-9]+
[18]	Color	::=	'#' AlphaOrDigits
[19]	AlphaOrDigits	::=	[a-fA-F0-9]+
[20]	Literal	::=	"" [^"]* "" "" "" [^']* "" ""
[21]	Operator	::=	OperatorName MultiplyOperator '+' '-'
[22]	OperatorName	::=	'mod' 'div'
[23]	MultiplyOperator	::=	'*'
[24]	Keyword	::=	'inherit'
[25]	FunctionName	::=	NCName
[26]	EnumerationToken	::=	NCName
[27]	AbsoluteUnitName	::=	'cm' 'mm' 'in' 'pt' 'pc' 'px'
[28]	RelativeUnitName	::=	'em'
[29]	ExprWhitespace	::=	S

5.9.12. Expression Value Conversions

Values that are the result of an expression evaluation may be converted into property value types. In some instances this is a simple verification of set membership (e.g., is the value a legal country code). In other cases, the value is expected to be a simple type like an integer and must be converted.

It is not necessary that all types be allowed to be converted. If the expression value cannot be converted to the necessary type for the property value, it is an error.

The following table indicates what conversions are allowed.

Type	Allowed Conversions	Constraints
NCName	<ul style="list-style-type: none"> Color, via the system-color() function. Enumeration value, as defined in the property definition. To a string literal 	The value may be checked against a legal set of values depending on the property.
AbsoluteNumeric	<ul style="list-style-type: none"> Integer, via the round() function. Color, as an RGB color value. 	If converting to an RGB color value, it must be a legal color value from the color space.
RelativeLength	<ul style="list-style-type: none"> To an AbsoluteLength 	

The specific conversion to be applied is property specific and can be found in the definition of each property.

5.9.13. Definitions of Units of Measure

The units of measure in this Recommendation have the following definitions:

Name	Definition
cm	See [ISO31]
mm	See [ISO31]
in	2.54cm
pt	1/72in
pc	12pt
px	See § 5.9.13.1 – Pixels on page 58
em	See § 5.9.7.2 – Relative Lengths on page 56

5.9.13.1. Pixels

XSL interprets a 'px' unit to be a request for the formatter to choose a device-dependent measurement that approximates viewing one pixel on a typical computer monitor. This interpretation is follows:

- The preferred definition of one 'px' is:
 - The actual distance covered by the largest integer number of device dots (the size of a device dot is measured as the distance between dot centers) that spans a distance less-than-or-equal-to the distance specified by the arc-span rule in <http://www.w3.org/TR/REC-CSS2/syndata.html#x39> (for print, desktop computer monitors, and hand-held devices viewed at normal viewing distances this is assumed to be 0.28mm [approximately 1/90]).
 - A minimum of the size of 1 device dot should be used.
 - This calculation is done separately in each axis, and may have a different value in each axis.
- However, implementors may instead simply pick a fixed conversion factor, treating 'px' as an

absolute unit of measurement (such as 1/92" or 1/72").

NOTE: Pixels should not be mixed with other absolute units in expressions as they may cause undesirable effects. Also, particular caution should be used with inherited property values that may have been specified using pixels.

If the User Agent chooses a measurement for a 'px' that does not match an integer number of device dots in each axis it may produce undesirable effects, such as:

- moiré patterns in scaled raster graphics
- unrenderable overlapping areas when the renderer rounds fonts or graphics sizes upward to its actual dot-size
- large spaces between areas when the renderer rounds fonts or graphics sizes downward to its actual dot-size
- unreadable results including unacceptably small text/layout (for example, a layout was done at 72 dpi [dots per inch], but the renderer assumed the result was already specified in device dots and renders it at 600 dpi).

Stylesheet authors should understand a pixel's actual size may vary from device to device:

- stylesheets utilizing 'px' units may not produce consistent results across different implementations or different output devices from a single implementation
- even if stylesheets are expressed entirely in 'px' units the results may vary on different devices

5.10. Core Function Library

5.10.1. Number Functions

Function: *numeric floor(numeric)*

The **floor** function returns the largest (closest to positive infinity) integer that is not greater than the argument. The numeric argument to this function must be of unit power zero.

NOTE: If it is necessary to use the **floor** function for a property where a unit power of one is expected, then an expression such as: "floor(1.4in div 1.0in)*1.0in" must be used. This applies to the ceiling, round, and other such functions where a unit power of zero is required.

Function: *numeric ceiling(numeric)*

The **ceiling** function returns the smallest (closest to negative infinity) integer that is not less than the argument. The numeric argument to this function must be of unit power zero.

Function: *numeric round(numeric)*

The **round** function returns the integer that is closest to the argument. If there are two such numbers, then the one that is closest to positive infinity is returned. The numeric argument to this function must be of unit power zero.

Function: *numeric min(numeric, numeric)*

The **min** function returns the minimum of the two numeric arguments. These arguments must have the same unit power.

Function: *numeric max(numeric, numeric)*

The **max** function returns the maximum of the two numeric arguments. These arguments must have the same unit power.

Function: *numeric abs(numeric)*

The **abs** function returns the absolute value of the numeric argument. That is, if the numeric argument

is negative, it returns the negation of the argument.

5.10.2. Color Functions

Function: *color* **rgb**(*numeric, numeric, numeric*)

The **rgb** function returns a specific color from the RGB color space. The parameters to this function must be numerics (real numbers) with a length power of zero.

Function: *color* **icc-color**(*numeric, numeric, numeric, NCName, numeric, numeric*)

The **icc-color** function returns a specific color from the ICC Color Profile. The color profile is specified by the name parameter (the fourth parameter). This color profile must have been declared in the fo:declarations formatting object using an fo:color-profile formatting object.

The first three parameters specify a fallback color from the sRGB color space. This color is used when the color profile is not available.

The color is specified by a sequence of one or more color values (real numbers) specified after the name parameter. These values are specific to the color profile

Function: *color* **system-color**(*NCName*)

The **system-color** function returns a system defined color with a given name.

5.10.3. Font Functions

Function: *object* **system-font**(*NCName, NCName?*)

The **system-font** functions returns a characteristic of a system font. The first argument is the name of the system font and the second argument, which is optional, names the property that specifies the characteristic. If the second argument is omitted, then the characteristic returned is the same as the name of the property to which the expression is being assigned.

For example, the expression "system-font(heading,font-size)" returns the font-size characteristic for the system font named "heading". This is equivalent to the property assignment 'font-size="system-font(heading)"'.

5.10.4. Property Value Functions

Function: *object* **inherited-property-value**(*NCName*)

The **inherited-property-value** function returns the inherited value of the property whose name matches the argument specified. It is an error if this property is not an inherited property.

The returned "inherited value" is the computed value of this property on this object's parent. In particular, given the following example:

```
<fo:list-block>
  ...
  <fo:list-item color="red">
    <fo:list-item-body background-color="green">
      <fo:block background-color="inherited-property-value(color)">
        </fo:block>
      </fo:list-item-body>
    </fo:list-item>
  </fo:list-block>
```

The background-color property on the fo:block is assigned the value "red" because the (computed, after inheritance) value of the color (not background-color) property on the fo:list-item-body that is the parent of fo:block is "red".

Function: *numeric label-end()*

The **label-end** function returns the calculated label-end value for lists. See the definition in § 7.26.3 – “provisional-label-separation” on page 320.

Function: *numeric body-start()*

The **body-start** function returns the calculated body-start value for lists. See the definition in § 7.26.4 – “provisional-distance-between-starts” on page 321.

NOTE: When this function is used outside of a list, it still returns a calculated value as specified.

Function: *object from-parent(NCName)*

The **from-parent** function returns a computed value of the property whose name matches the argument specified. The value returned is that for the parent of the formatting object for which the expression is evaluated. If there is no parent, the value returned is the initial value. If the argument specifies a shorthand property and if the expression only consists of the from-parent function with an argument matching the property being computed, it is interpreted as an expansion of the shorthand with each property into which the shorthand expands, each having a value of from-parent with an argument matching the property. It is an error if arguments matching a shorthand property are used in any other way.

Function: *object from-nearest-specified-value(NCName?)*

The **from-nearest-specified-value** function returns a computed value of the property whose name matches the argument specified. The value returned is that for the closest ancestor of the formatting object for which the expression is evaluated on which there is an assignment of the property in the XML result tree in the fo namespace. If there is no such ancestor, the value returned is the initial value. If the argument specifies a shorthand property and if the expression only consists of the from-nearest-specified-value function with an argument matching the property being computed, it is interpreted as an expansion of the shorthand with each property into which the shorthand expands, each having a value of from-nearest-specified-value with an argument matching the property. It is an error if arguments matching a shorthand property are used in any other way.

Function: *object from-table-column(NCName?)*

The **from-table-column** function returns the inherited value of the property whose name matches the argument specified, from the fo:table-column whose column-number matches the column for which this expression is evaluated and whose number-columns-spanned also matches any span. If there is no match for the number-columns-spanned, it is matched against a span of 1. If there is still no match, the initial value is returned. It is an error to use this function on formatting objects that are not an fo:table-cell or its descendants.

Function: *numeric proportional-column-width(numeric)*

The **proportional-column-width** function returns N units of proportional measure where N is the argument given to this function. The column widths are first determined ignoring the proportional measures. The difference between the table-width and the sum of the column widths is the available proportional width. One unit of proportional measure is the available proportional width divided by the sum of the proportional factors. It is an error to use this function on formatting objects other than an

fo:table-column. It is also an error to use this function if the fixed table layout is not used.

Function: *object merge-property-values(NCName?)*

The **merge-property-values** function returns a value of the property whose name matches the argument. The value returned is the specified value on the last fo:multi-property-set, of the parent fo:multi-properties, that applies to the User Agent state. If there is no such value, the computed value of the parent fo:multi-properties is returned.

NOTE: The test for applicability of a User Agent state is specified using the "active-state" property.

It is an error to use this function on formatting objects other than an fo:wrapper that is the child of an fo:multi-properties.

5.11. Property Datatypes

Certain property values are described in terms of compound datatypes, in terms of restrictions on permitted number values, or strings with particular semantics.

The compound datatypes, such as space, are represented in the result tree as multiple attributes. The names of these attributes consist of the property name, followed by a period, followed by the component name. For example a "space-before" property may be specified as:

```
space-before.minimum="2.0pt "
space-before.optimum="3.0pt "
space-before.maximum="4.0pt "
space-before.precedence="0"
space-before.conditionality="discard"
```

A short form of compound value specification may be used, in cases where the datatype has some <length> components and for the <keep> datatype. In the first case the specification consists of giving a <length> value to an attribute with a name matching a property name. Such a specification gives that value to each of the <length> components and the initial value to all the non-<length> components. For example:

```
space-before="4.0pt "
```

is equivalent to a specification of

```
space-before.minimum="4.0pt "
space-before.optimum="4.0pt "
space-before.maximum="4.0pt "
space-before.precedence="0"
space-before.conditionality="discard"
```

For the <keep> datatype the specification consists of giving a value that is valid for a component to an attribute with a name matching a property name. Such a specification gives that value to each of the components. For example:

```
keep-together="always"
```

is equivalent to a specification of

```
space-together.within-line="always"
space-together.within-columns="always"
space-together.within-page="always"
```

Short forms may be used together with complete forms; the complete forms have precedence over the

expansion of a short form. For example:

```
space-before="4.0pt"  
space-before.maximum="6.0pt"
```

is equivalent to a specification of

```
space-before.minimum="4.0pt"  
space-before.optimum="4.0pt"  
space-before.maximum="6.0pt"  
space-before.precedence="0"  
space-before.conditionality="discard"
```

Compound values of properties are inherited as a unit and not as individual components.

The following datatypes are defined:

<integer>

A signed integer value which consists of an optional '+' or '-' character followed by a sequence of digits. A property may define additional constraints on the value.

<number>

A signed real number which consists of an optional '+' or '-' character followed by a sequence of digits followed by an optional '.' character and sequence of digits. A property may define additional constraints on the value.

<length>

A signed length value where a 'length' is a real number plus a unit qualification. A property may define additional constraints on the value.

<length-range>

A compound datatype, with components: minimum, optimum, maximum. Each component is a <length>. A property may define additional constraints on the values.

<length-conditional>

A compound datatype, with components: length, conditionality. The length component is a <length>. The conditionality component is either "discard" or "retain". A property may define additional constraints on the values.

<length-bp-ip-direction>

A compound datatype, with components: block-progression-direction, and inline-progression-direction. Each component is a <length>. A property may define additional constraints on the values.

<space>

A compound datatype, with components: minimum, optimum, maximum, precedence, and conditionality. The minimum, optimum, and maximum components are <length>s. The precedence component is either "force" or an <integer>. The conditionality component is either "discard" or "retain".

<keep>

A compound datatype, with components: within-line, within-column, and within-page. The value of each component is either "auto", "always", or an <integer>.

<angle>

An <integer> representing an angle.

<percentage>

A signed real percentage which consists of an optional '+' or '-' character followed by a sequence of digits followed by an optional '.' character and sequence of digits followed by '%'. A property may define additional constraints on the value.

<character>

A single Unicode character.

<string>

A sequence of characters.

<name>

A string of characters representing a name. It must conform to the definition of an [NCName](#) in [\[W3C XML\]](#).

<family-name>

A string of characters identifying a font.

<color>

Either a string of characters representing a keyword or a color function defined in [§ 5.10.2 – Color Functions](#) on page 60. The list of keyword color names is: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow.

<country>

A string of characters conforming to an ISO 3166 country code.

<language>

A string of characters conforming to the ISO 639 3-letter code.

<script>

A string of characters conforming to an ISO 15924 script code.

<id>

A string of characters conforming to the XML NMTOKEN definition that is unique within the stylesheet.

<idref>

A string of characters conforming to the XML NMTOKEN definition that matches an ID property value used within the stylesheet.

<uri-specification>

A sequence of characters that is "url(", followed by optional whitespace, followed by an optional single quote (') or double quote (") character, followed by a URI-reference as defined in [\[RFC2396\]](#), followed by an optional single quote (') or double quote (") character, followed by optional whitespace, followed by ")". The two quote characters must be the same and must both be present or absent. If the URI-reference contains a single quote, the two quote characters must be present and be double quotes.

6. Formatting Objects

6.1. Introduction to Formatting Objects

The refined formatting object tree describes one or more intended presentations of the information within this tree. Formatting is the process which converts the description into a presentation. See [§ 3 – Introduction to Formatting](#) on page 10. The presentation is represented, abstractly, by an area tree, as defined in the area model. See [§ 4 – Area Model](#) on page 13. Each possible presentation is represented by one or more area trees in which the information in the refined formatting object tree is positioned on a two and one-half dimensional surface.

There are three kinds of formatting objects: (1) those that generate areas, (2) those that return areas, but do not generate them, and (3) those that are used in the generation of areas. The first and second kinds are typically called *flow objects*. The third kind is either a *layout object* or an *auxiliary object*. The kind of formatting object is indicated by the terminology used with the object. Formatting objects of the first kind are said to "generate one or more areas". Formatting objects of the second kind are said to "return one or more areas". Formatting objects of the first kind may both generate and return areas. Formatting objects of the third kind are "used in the generation of areas"; that is, they act like parameters to the generation process.

6.1.1. Definitions Common to Many Formatting Objects

This categorization leads to defining two traits which characterize the relationship between an area and the formatting objects which generate and return that area. These traits are *generated-by* and *returned-by*.

The value of the *generated-by* trait is a single formatting object. A formatting object *F* is defined to *generate* an area *A* if the semantics of *F* specify the generation of one or more areas and *A* is one of the areas thus generated, or is a substituted form of one of the areas thus generated, as specified in section [§ 4.7.2 – Line-building](#) on page 33.

In the case of substituted glyph-areas, the generating formatting object is deemed to be the formatting object which generated the glyph-area which comes first in the sequence of substituted glyph-areas. In the case of an inserted glyph-area (e.g., an automatically-generated hyphen) the generating formatting object is deemed to be the generating formatting object of the last glyph-area preceding the inserted glyph-area in the pre-order traversal of the area tree.

The value of the *returned-by* trait is a set of pairs, where each pair consists of a formatting object and a positive integer. The integer represents the position of the area in the ordering of all areas returned by the formatting object.

A formatting object *F* is defined to *return the sequence of areas* *A*, *B*, *C*, ... if the pair (*F*,1) is a member

of the *returned-by* trait of *A*, the pair (*F*,2) is a member of the *returned-by* trait of *B*, the pair (*F*,3) is a member of the *returned-by* trait of *C*, ...

If an area is a member of the sequence of areas returned by a formatting object, then either it was generated by the formatting object or it was a member of the sequence of areas returned by a child of that formatting object. Not all areas returned by a child of a formatting object need be returned by that formatting object. A formatting object may generate an area that has, as some of its children areas, areas returned by the children of that formatting object. These children (in the area tree) of the generated area are not returned by the formatting object to which they were returned.

A set of nodes in a tree is a *lineage* if:

- there is a node *N* in the set such that all the nodes in the set are ancestors of *N*, and
- for every node *N* in the set, if the set contains an ancestor of *N*, it also contains the parent of *N*.

The set of formatting objects that an area is returned by is a *lineage*.

Areas returned by a formatting object may be either *normal* or *out-of-line*. Normal areas represent areas in the "normal flow of text"; that is, they become area children of the areas generated by the formatting object to which they are returned. Normal areas have a *returned-by* lineage of size one. There is only one kind of normal area.

Out-of-line areas are areas used outside the normal flow of text either because they are absolutely positioned or they are part of a float or footnote. Out-of-line areas may have a *returned-by* lineage of size greater than one.

The *area-class* trait indicates which class, normal or out-of-line, an area belongs to. For out-of-line areas, it also indicates the subclass of out-of-line area. The values for this trait are: "xsl-normal", "xsl-absolute", "xsl-footnote", "xsl-side-float", or "xsl-before-float". An area is normal if and only if the value of the *area-class* trait is "xsl-normal"; otherwise, the area is an out-of-line area. (See section § 4.2.5 – [Stacking Constraints](#) on page 19.)

The areas *returned-by* a given formatting object are ordered as noted above. This ordering defines an ordering on the sub-sequence of areas that are of a given *area-class*, such as the sub-sequence of normal areas. An area *A* precedes an area *B* in the sub-sequence if and only if area *A* precedes area *B* in the areas *returned-by* the formatting objects.

A *reference-area chain* is defined as a sequence of reference-areas that is either generated by the same formatting object that is not a page-sequence formatting object, or that consists of the region reference-areas or main-reference-areas (see § 6.4.13 – [fo:region-body](#) on page 86) generated using region formatting objects assigned to the same flow (see § 6.4.1.4 – [Flows and Flow Mapping](#) on page 74). The reference-areas in the sequence are said to be "contained" by the reference-area chain, and they have the same ordering relative to each other in the sequence as they have in the area tree, using pre-order traversal order of the area tree.

6.2. Formatting Object Content

The content of a formatting object is described using XML content-model syntax. In some cases additional constraints, not expressible in XML content models, are given in prose.

The parameter entity, "%block;" in the content models below, contains the following formatting objects:

```
block
block-container
table-and-caption
```

table
list-block

The parameter entity, "%inline;" in the content models below, contains the following formatting objects:

bidi-override
character
external-graphic
instream-foreign-object
inline
inline-container
leader
page-number
page-number-citation
basic-link
multi-toggle

The following formatting objects are "neutral" containers and may be used anywhere where #PCDATA, %block;, or %inline; are allowed:

multi-switch
multi-properties
wrapper

The following "out-of-line" formatting objects may be used anywhere where #PCDATA, %block;, or %inline; are allowed (except as a descendant of any "out-of-line" formatting object):

float
footnote

6.3. Formatting Objects Summary

basic-link

The fo:basic-link is used for representing the start resource of a simple link.

bidi-override

The fo:bidi-override inline formatting object is used where it is necessary to override the default Unicode-bidirectionality algorithm direction for different (or nested) inline scripts in mixed-language documents.

block

The fo:block formatting object is commonly used for formatting paragraphs, titles, headlines, figure and table captions, etc.

block-container

The fo:block-container flow object is used to generate a block-level reference-area.

character

The fo:character flow object represents a character that is mapped to a glyph for presentation.

color-profile

Used to declare a color profile for a stylesheet.

conditional-page-master-reference

The fo:conditional-page-master-reference is used to identify a page-master that is to be used when the conditions on its use are satisfied.

declarations

Used to group global declarations for a stylesheet.

external-graphic

The fo:external-graphic flow object is used for a graphic where the graphics data resides outside of the XML result tree in the fo namespace.

float

The fo:float serves two purposes. It can be used so that during the normal placement of content, some related content is formatted into a separate area at beginning of the page (or of some following page) where it is available to be read without immediately intruding on the reader. Alternatively, it can be used when an area is intended to float to one side, with normal content flowing alongside.

flow

The content of the fo:flow formatting object is a sequence of flow objects that provides the flowing text content that is distributed into pages.

footnote

The fo:footnote is used to produce a footnote citation and the corresponding footnote.

footnote-body

The fo:footnote-body is used to generate the content of the footnote.

initial-property-set

The fo:initial-property-set specifies formatting properties for the first line of an fo:block.

inline

The fo:inline formatting object is commonly used for formatting a portion of text with a background or enclosing it in a border.

inline-container

The fo:inline-container flow object is used to generate an inline reference-area.

instream-foreign-object

The fo:instream-foreign-object flow object is used for an inline graphic or other "generic" object where the object data resides as descendants of the fo:instream-foreign-object.

layout-master-set

The fo:layout-master-set is a wrapper around all masters used in the document.

leader

The fo:leader formatting object is used to generate leaders consisting either of a rule or of a row

of a repeating character or cyclically repeating pattern of characters that may be used for connecting two text formatting objects.

list-block

The fo:list-block flow object is used to format a list.

list-item

The fo:list-item formatting object contains the label and the body of an item in a list.

list-item-body

The fo:list-item-body formatting object contains the content of the body of a list-item.

list-item-label

The fo:list-item-label formatting object contains the content of the label of a list-item; typically used to either enumerate, identify, or adorn the list-item's body.

marker

The fo:marker is used in conjunction with fo:retrieve-marker to produce running headers or footers.

multi-case

The fo:multi-case is used to contain (within an fo:multi-switch) each alternative sub-tree of formatting objects among which the parent fo:multi-switch will choose one to show and will hide the rest.

multi-properties

The fo:multi-properties is used to switch between two or more property sets that are associated with a given portion of content.

multi-property-set

The fo:multi-property-set is used to specify an alternative set of formatting properties that, dependent on a User Agent state, are applied to the content.

multi-switch

The fo:multi-switch wraps the specification of alternative sub-trees of formatting objects (each sub-tree being within an fo:multi-case), and controls the switching (activated via fo:multi-toggle) from one alternative to another.

multi-toggle

The fo:multi-toggle is used within an fo:multi-case to switch to another fo:multi-case.

page-number

The fo:page-number formatting object is used to represent the current page-number.

page-number-citation

The fo:page-number-citation is used to reference the page-number for the page containing the first normal area returned by the cited formatting object.

page-sequence

The fo:page-sequence formatting object is used to specify how to create a (sub-)sequence of pages within a document; for example, a chapter of a report. The content of these pages comes from flow children of the fo:page-sequence.

page-sequence-master

The fo:page-sequence-master specifies sequences of page-masters that are used when generating a sequence of pages.

region-after

This region defines a viewport that is located on the "after" side of fo:region-body region.

region-before

This region defines a viewport that is located on the "before" side of fo:region-body region.

region-body

This region specifies a viewport/reference pair that is located in the "center" of the fo:simple-page-master.

region-end

This region defines a viewport that is located on the "end" side of fo:region-body region.

region-start

This region defines a viewport that is located on the "start" side of fo:region-body region.

repeatable-page-master-alternatives

An fo:repeatable-page-master-alternatives specifies a sub-sequence consisting of repeated instances of a set of alternative page-masters. The number of repetitions may be bounded or potentially unbounded.

repeatable-page-master-reference

An fo:repeatable-page-master-reference specifies a sub-sequence consisting of repeated instances of a single page-master. The number of repetitions may be bounded or potentially unbounded.

retrieve-marker

The fo:retrieve-marker is used in conjunction with fo:marker to produce running headers or footers.

root

The fo:root node is the top node of an XSL result tree. This tree is composed of formatting objects.

simple-page-master

The fo:simple-page-master is used in the generation of pages and specifies the geometry of the page. The page may be subdivided into up to five regions.

single-page-master-reference

An fo:single-page-master-reference specifies a sub-sequence consisting of a single instance of a single page-master.

static-content

The fo:static-content formatting object holds a sequence or a tree of formatting objects that is to be presented in a single region or repeated in like-named regions on one or more pages in the page-sequence. Its common use is for repeating or running headers and footers.

table

The fo:table flow object is used for formatting the tabular material of a table.

table-and-caption

The fo:table-and-caption flow object is used for formatting a table together with its caption.

table-body

The fo:table-body formatting object is used to contain the content of the table body.

table-caption

The fo:table-caption formatting object is used to contain block-level formatting objects containing the caption for the table only when using the fo:table-and-caption.

table-cell

The fo:table-cell formatting object is used to group content to be placed in a table cell.

table-column

The fo:table-column formatting object specifies characteristics applicable to table cells that have the same column and span.

table-footer

The fo:table-footer formatting object is used to contain the content of the table footer.

table-header

The fo:table-header formatting object is used to contain the content of the table header.

table-row

The fo:table-row formatting object is used to group table-cells into rows.

title

The fo:title formatting object is used to associate a title with a given document. This title may be used by an interactive User Agent to identify the document. For example, the content of the fo:title can be formatted and displayed in a "title" window or in a "tool tip".

wrapper

The fo:wrapper formatting object is used to specify inherited properties for a group of formatting objects. It has no additional formatting semantics.

6.4. Declarations and Pagination and Layout Formatting Objects

6.4.1. Introduction

The root node of the formatting object tree must be an `fo:root` formatting object. The children of the `fo:root` formatting object are a single `fo:layout-master-set`, an optional `fo:declarations`, and a sequence of one or more `fo:page-sequences`. The `fo:layout-master-set` defines the geometry and sequencing of the pages; the children of the `fo:page-sequences`, which are called *flows* (contained in `fo:flow` and `fo:static-content`), provide the content that is distributed into the pages. The `fo:declarations` object is a wrapper for formatting objects whose content is to be used as a resource to the formatting process. The process of generating the pages is done automatically by the XSL processor formatting the result tree.

The children of the `fo:layout-master-set` are the pagination and layout specifications. The names of these specifications end in "-master". There are two types of pagination and layout specifications: `page-masters` and `page-sequence-masters`. `Page-masters` have the role of describing the intended subdivisions of a page and the geometry of these subdivisions. `Page-sequence-masters` have the role of describing the sequence of `page-masters` that will be used to generate pages during the formatting of an `fo:page-sequence`.

6.4.1.1. Page-sequence-masters

Each `fo:page-sequence-master` characterizes a set of possible sequences of `page-masters`. For any given `fo:page-sequence`, only one of the possible set of sequences will be used. The sequence that is used is any sequence that satisfies the constraints determined by the individual `page-masters`, the flows which generate pages from the `page-masters`, and the `fo:page-sequence-master` itself.

The `fo:page-sequence-master` is used to determine which `page-masters` are used and in which order. The children of the `fo:page-sequence-master` are a sequence of sub-sequence specifications. The `page-masters` in a sub-sequence may be specified by a reference to a single `page-master` or as a repetition of one or more `page-masters`. For example, a sequence might begin with several explicit `page-masters` and continue with a repetition of some other `page-master` (or `masters`).

The `fo:single-page-master-reference` is used to specify a sub-sequence consisting of a single `page-master`.

There are two ways to specify a sub-sequence that is a repetition. The `fo:repeatable-page-master-reference` specifies a repetition of a single `page-master`. The `fo:repeatable-page-master-alternatives` specifies the repetition of a set of `page-masters`. Which of the alternative `page-masters` is used at a given point in the sub-sequence is conditional and may depend on whether the page number is odd or even, is the first page, is the last page, or is blank. The "maximum-repeats" property on the repetition specification controls the number of repetitions. If this property is not specified, there is no limit on the number of repetitions.

6.4.1.2. Page-masters

A `page-master` is a master that is used to generate a *page*. A page is a viewport/reference pair in which the viewport-area is a child of the area tree root. A *page-viewport-area* is defined to be the viewport-area of a page, and a *page-area* is defined to be the unique child of a *page-viewport-area*.

The *page-viewport-area* is defined by the output medium; the *page-area* holds the page contents and has the effect of positioning the page contents on the output medium.

A single `page-master` may be used multiple times. Each time it is used it generates a single page; for

example, a page-master that is referenced from an `fo:repeatable-page-master-reference` will be used by the `fo:page-sequence` to generate one page for each occurrence of the reference in the specified sub-sequence.

NOTE: When pages are used with a User Agent such as a Web browser, it is common that the each document has only one page. The viewport used to view the page determines the size of the page. When pages are placed on non-interactive media, such as sheets of paper, pages correspond to one or more of the surfaces of the paper. The size of the paper determines the size of the page.

In this specification, there is only one kind of page-master, the `fo:simple-page-master`. Future versions of this specification may add additional kinds of page-masters.

An `fo:simple-page-master` has, as children, specifications for one or more regions.

A region specification is used as a master, the *region-master*, in generating viewport/reference pair consisting of a *region-viewport-area* and a *region-reference-area*. The region-viewport-area is always a child of a page-area generated using the parent of the region-master.

NOTE: The regions on the page are analogous to "frames" in an HTML document. Typically, at least one of these regions is of indefinite length in one of its dimensions. For languages with a `lr-tb` (or `rl-tb`) writing-mode, this region is typically of indefinite length in the top-to-bottom direction. The viewport represents the visible part of the frame. The flow assigned to the region is viewed by scrolling the region reference-area through the viewport.

Each region is defined by a region formatting object. Each region formatting object has a name and a definite position. In addition, the region's height or width is fixed and the other dimension may be either fixed or indefinite. For example, a region that is the body of a Web page may have indefinite height.

The specification of the region determines the size and position of region-viewport-areas generated using the region formatting object. The positioning of the viewport is relative to its page-area parent.

For version 1.0 of this recommendation, a page-master will consist of up to five regions: "region-body" and four other regions, one on each side of the body. To allow the side regions to correspond to the current writing-mode, these regions are named "region-before" (which corresponds to "header" in the "lr-tb" writing-mode), "region-after" (which corresponds to "footer" in the "lr-tb" writing-mode), "region-start" (which corresponds to a "left-sidebar" in the "lr-tb" writing-mode) and "region-end" (which corresponds to a "right-sidebar" in the "lr-tb" writing-mode). It is expected that a future version of the recommendation will introduce a mechanism that allows a page-master to contain an arbitrary number of arbitrarily sized and positioned regions.

Some types of region have conditional sub-regions associated with them, and the associated region-reference-areas are divided up by having child areas corresponding to the sub-regions, including a "main-reference-area" for the region. For region-masters to which the column-count property applies, the main-reference-area is further subdivided by having child-areas designated as "span-reference-areas" whose number depends upon the number of spans (i.e. block-areas with `span="all"`) occurring on the page. These in turn are subdivided by having child-areas designated as "normal-flow-reference-areas", whose number depends on the number of columns specified.

6.4.1.3. Page Generation

Pages are generated by the formatter's processing of `fo:page-sequences`. As noted above, each page is a viewport/reference pair in which the viewport-area is a child of the area tree root. Each page is generated using a page-master to define the region-viewport-areas and region-reference-areas that correspond to the regions specified by that page-master.

Each `fo:page-sequence` references either an `fo:page-sequence-master` or a `page-master`. If the reference is to a `page-master`, this is interpreted as if it were a reference to an `fo:page-sequence-master` that repeats the referenced `page-master` an unbounded number of times. An `fo:page-sequence` references a `page-master` if either the `fo:page-sequence` directly references the `page-master` via the "master-name" property or that property references an `fo:page-sequence-master` that references the `page-master`.

6.4.1.4. Flows and Flow Mapping

There are two kinds of *flows*: `fo:static-content` and `fo:flow`. An `fo:static-content` flow holds content, such as the text that goes into headers and footers, that is repeated on many of the pages. The `fo:flow` flow holds content that is distributed across a sequence of pages. The processing of the `fo:flow` flow is what determines how many pages are generated to hold the `fo:page-sequence`. The `fo:page-sequence-master` is used as the generator of the sequence of `page-masters` into which the flow children content is distributed.

The children of a flow are a sequence of block-level flow objects. Each flow has a name, and no two `fo:flow` or `fo:static-content` formatting objects in the same `page-sequence` may have the same name.

The assignment of flows to regions on a `page-master` is determined by a *flow-map*. The *flow-map* is an association between the flow children of the `fo:page-sequence` and regions defined within the `page-masters` referenced by that `fo:page-sequence`.

In version 1.0 of this recommendation, the *flow-map* is implicit. The "flow-name" property of a flow specifies to which region that flow is assigned. Each region has a "region-name" property. The implicit *flow-map* assigns a flow to the region that has the same name. In future versions of XSL, the *flow-map* is expected to become an explicit formatting object.

To avoid requiring users to generate region-names, the regions all have default values for the "region-name" property. The `region-body`, `region-before`, `region-after`, `region-start`, and `region-end` have the default names "xsl-region-body", "xsl-region-before", "xsl-region-after", "xsl-region-start", and "xsl-region-end", respectively.

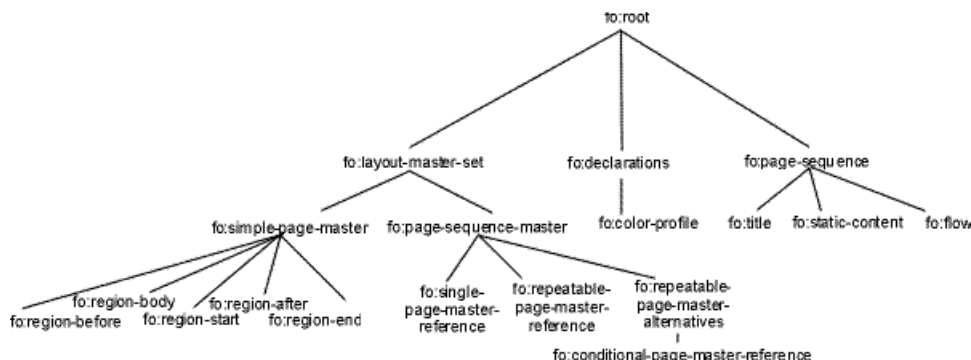
In addition, an `fo:static-content` formatting object may have a "region-name" property value of "xsl-before-float-separator" or "xsl-footnote-separator". If a conditional sub-region of the `region-body` is used to generate a reference-area on a particular page, the `fo:static-content` whose name corresponds to the conditional sub-region shall be formatted into the reference-area associated with the sub-region, as specified in section § 6.10.1.3 – Conditional Sub-Regions on page 151.

6.4.1.5. Constraints on Page Generation

The areas that are descendants of a `page-area` are constrained by the `page-master` used to generate the `page-area` and the flows that are assigned to the regions specified on the `page-master`. For `fo:flow` flows, the areas generated by the descendants of the flow are distributed across the pages in the sequence that were generated using `page-masters` having the region to which the flow is assigned. For `fo:static-content` flows, the processing of the flow is repeated for each page generated using a `page-master` having the region to which the flow is assigned with two exceptions: for a `fo:static-content` with a *flow-name* of `xsl-before-float-separator`, the processing is repeated only for those `page-reference-areas` which have descendant areas with an `area-class` of `xsl-before-float`, and for a `fo:static-content` with a *flow-name* of `xsl-footnote-separator`, the processing is repeated only for those `page-reference-areas` which have descendant areas with an `area-class` of `xsl-footnote`.

6.4.1.6. Pagination Tree Structure

The result tree structure is shown below.



Tree Representation of the Formatting Objects for Pagination

6.4.2. fo:root

Common Usage:

This is the top node of the formatting object tree. It holds an `fo:layout-master-set` formatting object (which holds all masters used in the document), an optional `fo:declarations`, and one or more `fo:page-sequence` objects. Each `fo:page-sequence` represents a sequence of pages that result from formatting the content children of the `fo:page-sequence`.

NOTE: A document can contain multiple `fo:page-sequences`. For example, each chapter of a document could be a separate `fo:page-sequence`; this would allow chapter-specific content, such as the chapter title, to be placed within a header or footer.

Areas:

Page-viewport-areas are returned by the `fo:page-sequence` children of the `fo:root` formatting object. The `fo:root` does not generate any areas.

Constraints:

The children of the root of the area tree consist solely of, and totally of, the page-viewport-areas returned by the `fo:page-sequence` children of the `fo:root`. The set of all areas returned by the `fo:page-sequence` children is *properly ordered*. (See Section § 4.7.1 – General Ordering Constraints on page 32.)

Contents:

`(layout-master-set , declarations? , page-sequence+)`

6.4.3. fo:declarations

Common Usage:

The `fo:declarations` formatting object is used to group global declarations for a stylesheet.

Areas:

The `fo:declarations` formatting object does not generate or return any areas.

Constraints:

None.

Contents:

([color-profile](#))⁺

The fo:declarations flow object may have additional child elements in a non-XSL namespace. Their presence does not, however, change the semantics of the XSL namespace objects and properties. The permitted structure of these non-XSL namespace elements is defined for their namespace(s).

6.4.4. fo:color-profile

Common Usage:

The fo:color-profile formatting object is used to declare an ICC Color Profile for a stylesheet. The color-profile is referenced again via the name specified in the "color-profile-name" property.

The color-profile is identified by the URI specified in the "src" property value. This URI may identify an internally recognized color-profile or it may point to a ICC Color Profile encoding that should be loaded and interpreted.

When the color-profile is referenced (e.g., via the [icc-color](#) function § 5.10.2 – Color Functions on page 60), the following rules are used:

1. If the color-profile is available, the color value identified from the color-profile should be used.
2. If the color-profile is not available, the sRGB fallback must be used.

Areas:

The fo:color-profile formatting object does not generate or return any areas.

Constraints:

None.

Contents:

EMPTY

The following properties apply to this formatting object:

- “src” — § 7.26.7 on page 322
- “color-profile-name” — § 7.15.2 on page 258
- “rendering-intent” — § 7.15.3 on page 258

6.4.5. fo:page-sequence

Common Usage:

The fo:page-sequence formatting object is used to specify how to create a (sub-)sequence of pages within a document; for example, a chapter of a report. The content of these pages comes from flow children of the fo:page-sequence. The layout of these pages comes from the fo:page-sequence-master or page-master referenced by the *master-name* trait on the fo:page-sequence. The sequence of areas returned by each of the flow-object children of the fo:page-sequence are made descendants of the generated pages as described below.

Areas:

The fo:page-sequence formatting object generates a sequence of viewport/reference pairs, and returns the page-viewport-areas. For each page-reference-area, and each region specified in the page-master used to generate that page-reference-area, the fo:page-sequence object also generates the viewport/reference pair for the occurrence of that region in that page-reference-area, and may generate a before-float-reference-area, footnote-reference-area, and main-reference-area, and one or more normal-sequence-reference-areas. The generation of these further areas is described in the descriptions of the fo:simple-page-master and region-masters.

All areas generated by an fo:page-sequence have area-class "xsl-absolute".

Constraints:

Each page-viewport-area/page-reference-area pair is generated using a page-master that satisfies the constraints of the page-sequence-master identified by the *master-name* trait of the fo:page-sequence or a page-master that was directly identified by the *master-name* trait. The region-viewport-area children of such a page-reference-area must correspond to the regions that are children of that page-master.

The areas generated by the fo:page-sequence have as their descendants the areas returned by the flows that are children of the fo:page-sequence.

The areas returned to the fo:page-sequence by a flow must satisfy four types of constraints:

- *Completeness.* All areas returned by formatting object descendants of the flow children of the fo:page-sequence become descendants of areas generated by the fo:page-sequence, with the exception of glyph-areas subject to deletion or substitution as in Sections [Line-building](#) — § 4.7.2 on page 33 and [Inline-building](#) — § 4.7.3 on page 34.
- *Flow-map association.* All areas returned by flow children of the fo:page-sequence become descendants of region-reference-areas generated from column-areas associated to the flow by the flow-map in effect, except for areas returned from a fo:static-content with a *flow-name* of xsl-before-float-separator or xsl-footnote-separator.

Areas returned from an fo:static-content with a *flow-name* of xsl-before-float-separator become children of the before-float-reference-area of an area associated to an fo:region-body, following all sibling areas of area-class xsl-before-float. Areas returned from an fo:static-content with a *flow-name* of xsl-footnote-separator become children of the footnote-reference-area of an area associated to an fo:region-body, preceding all sibling areas of area-class xsl-footnote.

- *Area-class association.* Areas returned by flow children of an fo:page-sequence are distributed as follows: all areas of area-class xsl-footnote must be descendants of a footnote-reference-area; areas of area-class xsl-before-float must be descendants of a before-float-reference-area; all other areas (including normal areas) must be descendants of a main-reference-area for a region.
- *Stacking.* The stackable areas of a given class returned by children of each flow are properly stacked within the appropriate reference-area, as described above.

The default ordering constraint of section [§ 4.7.1 – General Ordering Constraints](#) on page 32 does not apply to the fo:page-sequence. The default ordering constraints apply to the child fo:flow objects; special ordering constraints apply to the child fo:static-content objects.

Contents:

([title?](#), [static-content*](#), [flow](#))

The following properties apply to this formatting object:

- “country” — § 7.8.1 on page 202
- “format” — § 7.22.1 on page 287
- “language” — § 7.8.2 on page 202
- “letter-value” — § 7.22.4 on page 288
- “grouping-separator” — § 7.22.2 on page 287
- “grouping-size” — § 7.22.3 on page 287
- “id” — § 7.26.2 on page 320
- “initial-page-number” — § 7.23.7 on page 291
- “force-page-count” — § 7.23.6 on page 290
- “master-name” — § 7.23.8 on page 292

6.4.6. fo:layout-master-set

Common Usage:

The fo:layout-master-set is a wrapper around all masters used in the document. This includes page-sequence-masters, page-masters, and region-masters.

Areas:

The fo:layout-master-set formatting object generates no area directly. The masters that are the children of the fo:layout-master-set are used by the fo:page-sequence to generate pages.

Constraints:

The value of the *master-name* trait on each child of the fo:layout-master-set must be unique within the set.

Contents:

([simple-page-master](#) | [page-sequence-master](#)) +

6.4.7. fo:page-sequence-master

Common Usage:

The fo:page-sequence-master is used to specify the constraints on and the order in which a given set of page-masters will be used in generating a sequence of pages. Pages are automatically generated when the fo:page-sequence-master is used in formatting an fo:page-sequence.

NOTE: There are several ways of specifying a potential sequence of pages. One can specify a sequence of references to particular page-masters. This yields a bounded sequence of potential pages. Alternatively, one can specify a repeating sub-sequence of one or more page-masters. This sub-sequence can be bounded or unbounded. Finally one can intermix the two kinds of sub-sequence-specifiers.

Areas:

The fo:page-sequence-master formatting object generates no area directly. It is used by the fo:page-sequence formatting object to generate pages.

Constraints:

The children of the fo:page-sequence-master are a sequence of *sub-sequence-specifiers*. A page-sequence satisfies the constraint determined by an fo:page-sequence-master if (a) it can be partitioned into sub-sequences of pages that map one-to-one, in order, to the sub-sequence of

sub-sequence-specifiers that are the children of the fo:page-sequence-master and, (b) for each sub-sequence of pages, that sub-sequence satisfies the constraints of the corresponding sub-sequence-specifier. Note that the mapping of sub-sequences of the sequence of pages to sub-sequence-specifiers need not be onto; that is, the sequence of sub-sequences of pages can be shorter than the sequence of sub-sequence-specifiers.

It is an error if the entire sequence of sub-sequence-specifiers children is exhausted while some areas returned by an fo:flow are not placed. Implementations may recover, if possible, by re-using the sub-sequence-specifier that was last used to generate a page.

Contents:

(single-page-master-reference|repeatable-page-master-reference|repeatable-page-master-alternatives)+

The following properties apply to this formatting object:

- “master-name” — § 7.23.8 on page 292

6.4.8. fo:single-page-master-reference

Common Usage:

An fo:single-page-master-reference is the simplest sub-sequence-specifier. It specifies a sub-sequence consisting of a single instance of a single page-master. It is used to specify the use of a particular page-master at a given point in the sequence of pages that would be generated using the fo:page-sequence-master that is the parent of the fo:single-page-master-reference.

Areas:

The fo:single-page-master-reference formatting object generates no area directly. It is used by the fo:page-sequence formatting object to generate pages.

Constraints:

The fo:single-page-master-reference has a reference to the fo:simple-page-master which has the same master-name as the *master-name* trait on the fo:single-page-master-reference.

The sub-sequence of pages mapped to this sub-sequence-specifier satisfies the constraints of this sub-sequence-specifier if (a) the sub-sequence of pages consists of a single page and (b) that page is constrained to have been generated using the fo:simple-page-master referenced by the fo:single-page-master-reference.

Contents:

EMPTY

The following properties apply to this formatting object:

- “master-name” — § 7.23.8 on page 292

6.4.9. fo:repeatable-page-master-reference

Common Usage:

An fo:repeatable-page-master-reference is the next simplest sub-sequence-specifier. It specifies a sub-sequence consisting of repeated instances of a single page-master. The number of repetitions may be bounded or potentially unbounded.

Areas:

The fo:repeatable-page-master-reference formatting object generates no area directly. It is used by the fo:page-sequence formatting object to generate pages.

Constraints:

The fo:repeatable-page-master-reference has a reference to the fo:simple-page-master which has the same master-name as the *master-name* trait on the fo:repeatable-page-master-reference.

The sub-sequence of pages mapped to this sub-sequence-specifier satisfies the constraints of this sub-sequence-specifier if (a) the sub-sequence of pages consists of zero or more pages, (b) each page is generated using the fo:simple-page-master referenced by the fo:repeatable-page-master-reference, and (c) length of the sub-sequence is less than or equal to the value of *maximum-repeats*.

If no region-master child of the fo:repeatable-page-master has a region-name associated to any flow in an fo:page-sequence, then the sub-sequence is constrained to have length zero.

Contents:

EMPTY

The following properties apply to this formatting object:

- “[master-name](#)” — § 7.23.8 on page 292
- “[maximum-repeats](#)” — § 7.23.9 on page 293

6.4.10. fo:repeatable-page-master-alternatives

Common Usage:

The fo:repeatable-page-master-alternatives formatting object is the most complex sub-sequence-specifier. It specifies a sub-sequence consisting of repeated instances of a set of alternative page-masters. The number of repetitions may be bounded or potentially unbounded. Which of the alternative page-masters is used at any point in the sequence depends on the evaluation of a condition on the use of the alternative. Typical conditions include, testing whether the page which is generated using the alternative is the first or last page in a page-sequence or is the page blank. The full set of conditions allows different page-masters to be used for the first page, for odd and even pages, for blank pages.

NOTE: Because the conditions are tested in order from the beginning of the sequence of children, the last alternative in the sequence usually has a condition that is always true and this alternative references the page-master that is used for all pages that do not receive some specialized layout.

Areas:

The fo:repeatable-page-master-alternatives formatting object generates no area directly. This formatting object is used by the fo:page-sequence formatting object to generate pages.

Constraints:

The children of the fo:repeatable-page-master-alternatives are fo:conditional-page-master-references. These children will be called *alternatives*.

The sub-sequence of pages mapped to this sub-sequence-specifier satisfies the constraints of this sub-sequence-specifier if (a) the sub-sequence of pages consist of zero or more pages, (b) each page is generated using the fo:simple-page-master referenced by the one of the alternatives that are the children

of the `fo:repeatable-page-master-alternatives`, (c) the conditions on that alternative are `true`, (d) that alternative is the first alternative in the sequence of children for which all the conditions are `true`, and (e) the length of the sub-sequence is less than or equal to the value of *maximum-repeats*.

Contents:

(`conditional-page-master-reference`+))

The following properties apply to this formatting object:

- “`maximum-repeats`” — § 7.23.9 on page 293

6.4.11. `fo:conditional-page-master-reference`

Common Usage:

The `fo:conditional-page-master-reference` is used to identify a page-master that is to be used when the conditions on its use are satisfied. This allows different page-masters to be used, for example, for even and odd pages, for the first page in a page-sequence, or for blank pages. This usage is typical in chapters of a book or report where the first page has a different layout than the rest of the chapter and the headings and footings on even and odd pages may be different as well.

Areas:

The `fo:conditional-page-master-reference` formatting object generates no area directly. It is used by the `fo:page-sequence` formatting object to generate pages.

Constraints:

The `fo:conditional-page-master-reference` has a reference to the `fo:simple-page-master` which has the same *master-name* as the *master-name* trait on the `fo:conditional-page-master-reference`.

There are three traits, *page-position*, *odd-or-even*, and *blank-or-not-blank* that specify the sub-conditions on the use of the referenced page-master. All three sub-conditions must be `true` for the condition on the `fo:conditional-page-master-reference` to be `true`. Since the properties from which these traits are derived are not inherited and the initial value of all the properties makes the corresponding sub-condition `true`, this really means that the subset of traits that are derived from properties with specified values must make the corresponding sub-condition `true`.

The sub-condition corresponding to the *page-position* trait is `true` if the page generated using the `fo:conditional-page-master-reference` has the specified position in the sequence of pages generated by the referencing page-sequence; namely, "first", "last", "rest" (not first) or "any" (all of the previous). The *referencing page-sequence* is the `fo:page-sequence` that referenced the `fo:page-sequence-master` from which this `fo:conditional-page-master-reference` is a descendant.

The sub-condition corresponding to the *odd-or-even* trait is `true` if the value of the *odd-or-even* trait is "any" or if the value matches the parity of the page number of the page generated using the `fo:conditional-page-master-reference`.

The sub-condition corresponding to the *blank-or-not-blank* trait is `true`, if (1) the value of the trait is "not-blank" and the page generated using the `fo:conditional-page-master-reference` has areas generated by descendants of the `fo:flow` formatting object; if (2) the value of the trait is "blank" and the page generated using the `fo:conditional-page-master-reference` is such that there are no areas from the `fo:flow` to be put on that page (e.g., (a) to maintain proper page parity due to (i) a break-after or break-before value of "even-page" or "odd-page" or (ii) at the start or end of the page-sequence or (b) because the constraints on the areas generated by descendants of the `fo:flow` formatting object would not be satisfied

if they were descendant from this page); or if (3) the value of the trait is "any".

NOTE: If any page-master referenced from a conditional-page-master-reference with `blank-or-not-blank="true"` provides a region in which to put `fo:flow` content, no content is put in that region.

Contents:

EMPTY

The following properties apply to this formatting object:

- “[master-name](#)” — § 7.23.8 on page 292
- “[page-position](#)” — § 7.23.12 on page 295
- “[odd-or-even](#)” — § 7.23.10 on page 294
- “[blank-or-not-blank](#)” — § 7.23.1 on page 288

6.4.12. `fo:simple-page-master`

Common Usage:

The `fo:simple-page-master` is used in the generation of pages and specifies the geometry of the page. The page may be subdivided into up to five regions: `region-body`, `region-before`, `region-after`, `region-start`, and `region-end`.

NOTE: For example, if the *writing-mode* of the `fo:simple-page-master` is “`lr-tb`”, then these regions correspond to the body of a document, the header, the footer, the left sidebar, and the right sidebar.

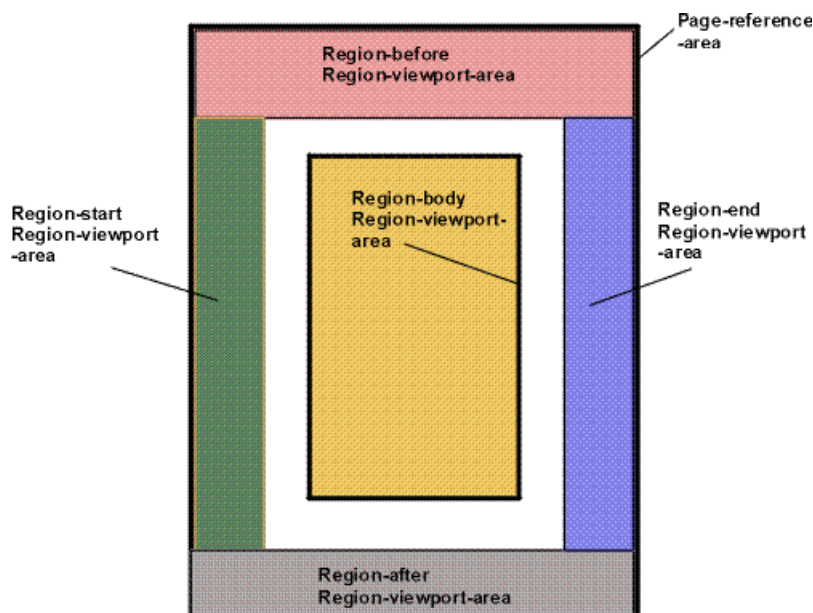
NOTE: The `simple-page-master` is intended for systems that wish to provide a simple page layout facility. Future versions of this recommendation will support more complex page layouts constructed using the `fo:page-master` formatting object.

Areas:

The `fo:simple-page-master` formatting object generates no area directly. It is used in the generation of pages by an `fo:page-sequence`.

When the `fo:simple-page-master` is used to generate a page, a viewport/reference pair is generated, consisting of a `page-viewport-area` and a `page-reference-area`. The `page-viewport-area` represents the physical bounds of the output medium. The `page-reference-area` represents the portion of the page on which content is intended to appear; that is, the area inside the page margins.

In addition, when the `fo:simple-page-master` is used to generate a page, viewport/reference pairs that correspond to the regions that are the children of the `fo:simple-page-master` are also generated. (See the formatting object specifications for the five regions (§ 6.4.13 – [fo:region-body](#) on page 86, § 6.4.14 – [fo:region-before](#) on page 89, § 6.4.15 – [fo:region-after](#) on page 90, § 6.4.16 – [fo:region-start](#) on page 91, and § 6.4.17 – [fo:region-end](#) on page 93) for the details on the generation of these areas.)



Region-viewport-areas

The spacing between the outer four regions and the fo:region-body is determined by subtracting the relevant *extent* trait on each outer region from the "margin-x" property on the fo:region-body.

Trait Derivation:

In version 1.0 of this recommendation, borders and padding are not allowed with a page-reference-area. The remaining traits on the page-reference-area are set according to normal rules for determining the values of traits.

Constraints:

When a page-master is used in the generation of a page, the *height* and *width* of the content-rectangle of the page-viewport-area are determined using the computed values of the "page-height" and "page-width" properties. If the "page-height" and "page-width" properties have explicit values, they are used to set the corresponding *height* and *width* trait on the page-viewport-area. If the "page-height" and/or "page-width" are set to "auto", the size of the page-viewport-area will be determined from the size of the media. If the media has a fixed size, then the size of the media is used. In the case of continuous media, the size of the User Agent window is used.

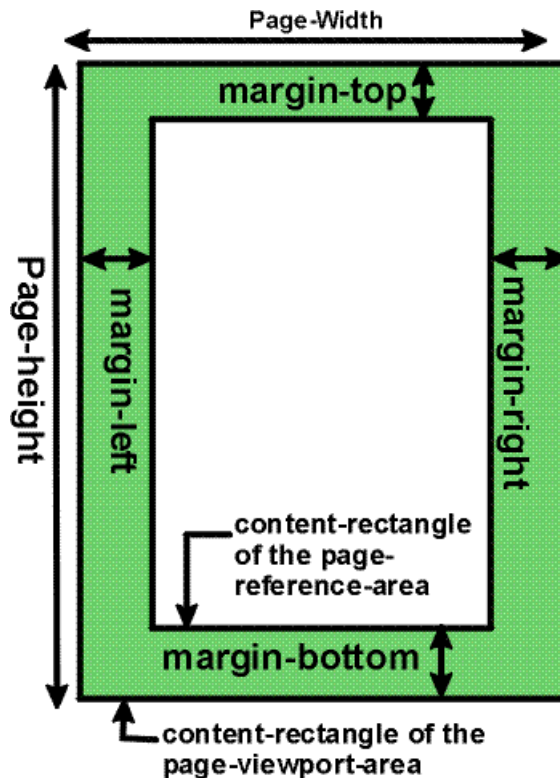
NOTE: The above size calculations are intended to match the handling of a frameset in a browser window when the media is continuous and to match pages when the media is paged.

A User Agent may provide a way to declare the media for which formatting is to be done. This may be different from the media on which the formatted result is viewed. For example, a browser User Agent may be used to preview pages that are formatted for sheet media. In that case, the size calculation is based on the media for which formatting is done rather than the media being currently used.

The traits derived from the margin properties determine the size and position of the content-rectangle of the page-viewport-area. The traits derived from the "margin-top", "margin-bottom", "margin-left" and "margin-right" properties are used to indent the page-reference-area content-rectangle from the corresponding edge of the content-rectangle of the page-viewport-area. Here "top", "bottom", "left" and "right" are determined by the computed values of the "page-height" and "page-width" properties. For sheet media, these values determine the orientation of the sheet; "page-height" is measured from "top"

to "bottom". For display media, the display window is always upright; the top of the display screen is "top".

NOTE: The reference points for the page-viewport-area content-rectangle are in terms of the "top", "bottom", "left", and "right" rather than "before-edge", "after-edge", "start-edge", and "end-edge" because users see the media relative to its orientation and not relative to the *writing-mode* currently in use.

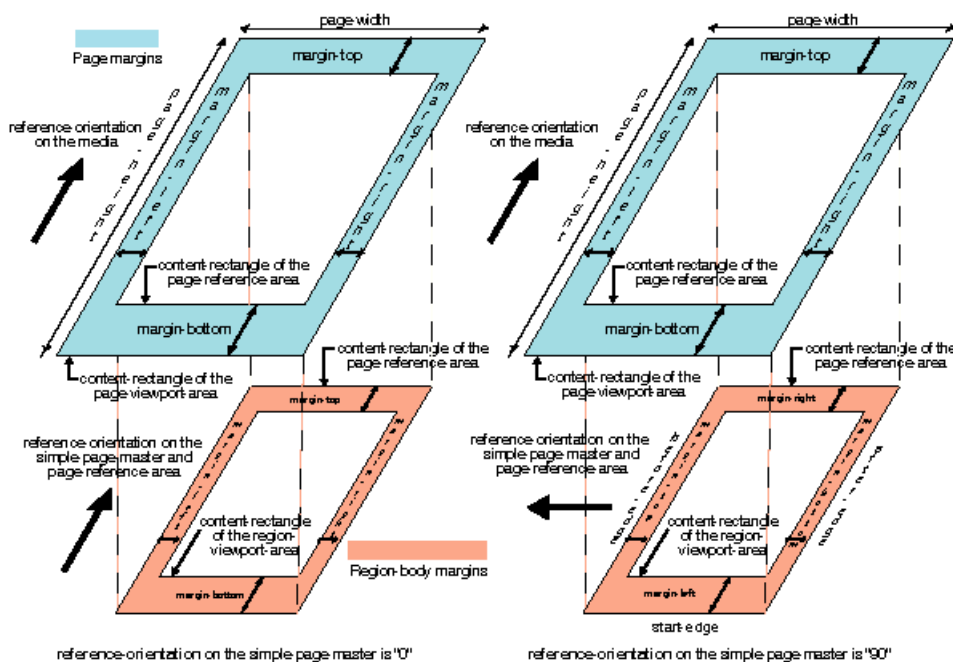


The value of the *page-number* trait on the first page returned by the `fo:page-sequence` is constrained to equal the value of the *initial-page-number* trait. The value of the *page-number* trait on subsequent pages is constrained to be one greater than the value on the immediately preceding page.

The *format*, *letter-value*, *grouping-separator*, *grouping-size*, *country*, and *language* traits are used to format the number into a string form, as specified in XSLT, section 7.7.1. This formatted number is used as the value of the `fo:page-number` flow object.

Constraints applicable to regions:

There are a number of constraints that apply to all the regions that are specified within a given `fo:simple-page-master`.



If the block-progression-dimension of the properly stacked region-reference-area is greater than the block-progression-dimension of the region-viewport-area that is its parent, then the constraints on the relationship between the region-viewport-area and the region-reference-area depend on values of the *overflow* trait on the region-master and the kind of flow assigned to the region.

If the flow assigned to the corresponding region is an *fo:static-content* flow object, then there is no constraint on the block-progression-dimension of the region-reference-area.

If the flow assigned to the corresponding region is an *fo:flow* formatting object, then

- If the value of the *overflow* trait is *scroll*, then there is no constraint on the block-progression-dimension of the region-reference-area.
- If the value of the *overflow* trait is *hidden*, *paginate*, *error-if-overflow*, or *visible*, then the block-progression-dimension of the region-reference-area is constrained to be no greater than the block-progression-dimension of the region-viewport-area.

The block-progression-dimension of a region-area that corresponds to a given region-master depends on the value of the *overflow* trait on that region-master.

Contents:

(*region-body*, *region-before?*, *region-after?*, *region-start?*, *region-end?*)

The following properties apply to this formatting object:

- [Common Margin Properties-Block](#) — § 7.9 on page 206
- “*master-name*” — § 7.23.8 on page 292
- “*page-height*” — § 7.23.11 on page 294
- “*page-width*” — § 7.23.13 on page 296
- “*reference-orientation*” — § 7.18.3 on page 270
- “*writing-mode*” — § 7.25.7 on page 317

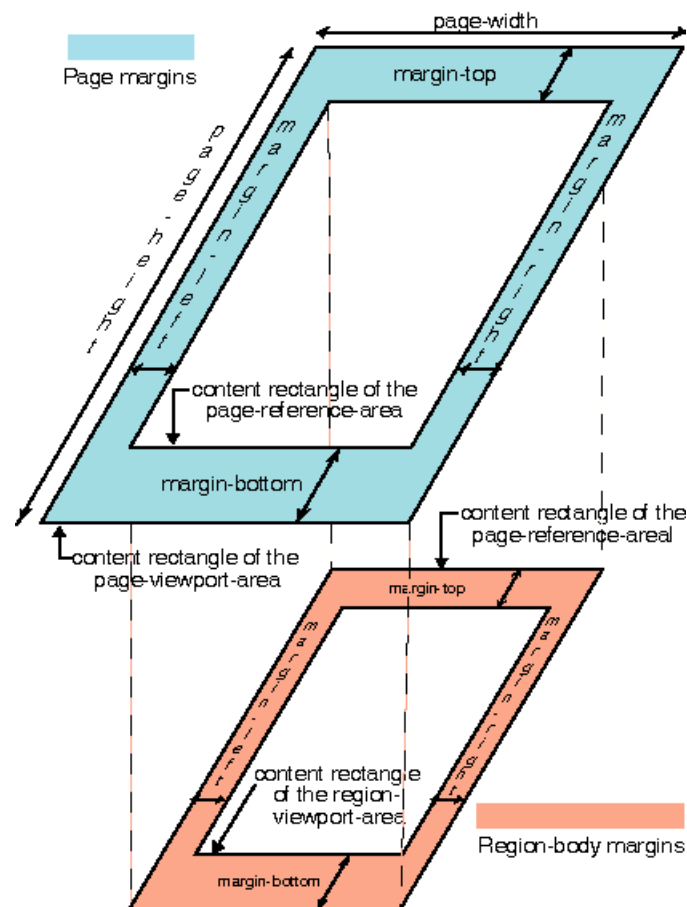
6.4.13. fo:region-body

Common Usage:

Used in constructing a simple-page-master. This region specifies a viewport/reference pair that is located in the "center" of the fo:simple-page-master. The *overflow* trait controls how much of the underlying region-reference-area is visible; that is, whether the region-reference-area is clipped by its parent region-viewport-area.

NOTE: Typically, for paged media, the areas returned by the fo:flow formatting object in a fo:page-sequence are made to be descendants of a sequence of region-reference-areas that correspond to the region-body. These region-reference-areas are all area descendants of page-areas for which the page-master included an fo:region-body. If the fo:flow flow is assigned to some other region, then the areas returned by the fo:flow are constrained to be descendants of region-reference-areas generated using the assigned region-master.

NOTE: The body region should be sized and positioned within the fo:simple-page-master so that there is room for the areas returned by the flow that is assigned to the fo:region-body and for any desired side regions, that is, fo:region-before, fo:region-after, fo:region-start and fo:region-end's that are to be placed on the same page. These side regions are positioned within the content-rectangle of the page-reference-area. The margins on the fo:region-body are used to position the region-viewport-area for the fo:region-body and to leave space for the other regions that surround the fo:region-body.



The spacing between the last four regions and the fo:region-body is determined by subtracting the relevant *extent* trait on the side regions from the trait that corresponds to the "margin-x" property on the fo:region-body.

The fo:region-body may be also be used to provide multiple columns. When the *column-count* trait is

greater than one, then the region-body will be subdivided into multiple columns.

Areas:

The `fo:region-body` formatting object is used to generate one region-viewport-area and one region-reference-area whenever an `fo:simple-page-master` that has an `fo:region-body` as a child is used to generate a page. A scrolling mechanism shall be provided, in an implementation-defined manner, if the value of the *overflow* trait is "scroll".

The position and size of the region-viewport-area is specified relative to the content-rectangle of the page-reference-area generated by `fo:simple-page-master`. The content-rectangle of the region-viewport-area is indented from the content-rectangle of the page-reference-area by the values of the "margin-top", "margin-bottom", "margin-left" and "margin-right" properties. In version 1.0 of this recommendation, the values of the *padding* and *border-width* traits must be "0".

The region-reference-area generated using an `fo:region-body` is the child of the region-viewport-area. The *reference-orientation* trait of the `fo:region-body` is used to orient the coordinate system of the region-reference-area generated by the `fo:region-body` relative to the coordinate system of the page-reference-area generated by `fo:simple-page-master` (and, therefore, relative to the viewport positioned in that latter coordinate system).

In addition to the viewport/reference pair, when the region-body is used to generate areas, at least one and up to three additional reference-areas are generated. These reference-areas are the optional *before-float-reference-area*, the optional *footnote-reference-area*, and the *main-reference-area*. The latter reference-area comprises the space left after space is borrowed for the other two reference-areas. The main-reference-area has no padding, border, or space associated with it.

NOTE: If there is no before-float-reference-area or footnote-reference-area child of the region-reference-area, then the content-rectangle of the main-reference-area is coterminous with the content-rectangle of the region-reference-area.

The main-reference-area has as its children a sequence of *span-reference-areas*. These are reference-area block-areas with zero border and padding, whose inline-progression-dimension is equal to that of the main-reference-area, and which are normally stacked within the main-reference-area.

Each span-reference-area has one or more reference-area children, designated as *normal-flow-reference-areas*. The number and placement of the children of a span-reference-area depends on the *column-count* trait of the span-reference-area. In turn, the formatter must generate sufficiently many of these span-reference-areas, and so set their *column-count* traits, that block-areas returned from the `fo:flow` with a *span* of "all" are children of span-reference-areas with *column-count* equal to 1, and block-areas returned from the `fo:flow` with a *span* of "none" are children of span-reference-areas with *column-count* equal to the refined value of the column-count property of the associated region-reference-area.

For each span-reference-area, the number *N* of normal-flow-reference-area children is equal to the value of the *column-count* trait.

It is an error to specify a *column-count* other than 1 if the "overflow" property has the value "scroll". An implementation may recover by behaving as if "1" had been specified.

The inline-progression-dimension of each of these normal-flow-reference-areas is determined by subtracting (*N*-1) times the column-gap trait from the inline-progression-dimension of the main-reference-area and dividing that result by *N*. Using "body-in-size" for the name of the inline-progression-dimension of the span-reference-area and "column-in-size" for the name of the size

of the normal-flow-reference-areas in the inline-progression-direction, the formula is:

$$\text{column-in-size} = (\text{body-in-size} - (N - 1) * \text{column-gap}) / N$$

The block-progression-dimension of the normal-flow-reference-areas is the same as that of the parent span-reference-area.

NOTE: As noted above, the block-progression-dimension of the span-reference-area may be less than the size of the region-reference-area if a before-float-reference-area or footnote-reference-area are present, or if there is more than one span-reference-area child of the main-reference-area.

The normal-flow-reference-areas are positioned within the span-reference-area as follows: The first column is positioned with the before-edge and start-edge of its content-rectangle coincident with the before-edge and start-edge of the content-rectangle of the span-reference-area. The content-rectangle of the J th normal-flow-reference-area child of the span-reference-area is positioned with its before-edge coincident with the before-edge of the content-rectangle of the span-reference-area and with its start-edge at $((J-1) * (\text{column-in-size} + \text{column-gap}))$ in the inline-progression-direction. This results in the end-edge of the content-rectangle of the N th normal-flow-reference-area being coincident with the end-edge of the content-rectangle of the span-reference-area.

NOTE: If the *writing-mode* is "rl-tb", the above description means that the columns are ordered from right-to-left as would be expected. This follows because the start-edge is on the right in an "rl-tb" writing-mode.

All areas generated by using the fo:region-body are of area-class "xsl-absolute".

Trait Derivation:

The *reference-orientation* of the region-viewport-area is taken from the value of the *reference-orientation* trait on the region-master which specifies the region. *reference-orientation* of the region-reference-area is set to "0" and is, therefore, the same as the orientation established by the region-viewport-area.

The remaining traits on the region-viewport-area and region-reference-area are set according to normal rules for determining the values of traits.

Constraints:

The constraints applicable to all regions (see § 6.4.12 – fo:simple-page-master on page 82) all apply.

The inline-progression-dimension of the region-viewport-area is determined by the inline-progression-dimension of the content-rectangle of the page-reference-area minus the values of the *start-indent* and *end-indent* traits of the region-master. The start-edge and end-edge of the content-rectangle of the region-viewport-area are determined by the *reference-orientation* trait on the page-master.

The block-progression-dimension of the region-viewport-area is determined by the block-progression-dimension of the content-rectangle for the page-reference-area minus the values of the *space-before* and *space-after* traits of the region-master. The before-edge and after-edge of the content-rectangle of the region-viewport-area are determined by the *reference-orientation* trait on the page-master.

The values of the *space-before* and *start-indent* traits are used to position the region-viewport-area relative to the before-edge and start-edge of the content-rectangle of the page-reference-area.

The constraints on the size and position of the region-reference-area generated using the fo:region-body are covered in the "Constraints applicable to regions" section of § 6.4.12 – fo:simple-page-master on

page 82.

Contents:

EMPTY

The following properties apply to this formatting object:

- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Block](#) — § 7.9 on page 206
- “clip” — § 7.18.1 on page 268
- “column-count” — § 7.23.2 on page 289
- “column-gap” — § 7.23.3 on page 289
- “display-align” — § 7.11.4 on page 226
- “overflow” — § 7.18.2 on page 269
- “region-name” — § 7.23.15 on page 297
- “reference-orientation” — § 7.18.3 on page 270
- “writing-mode” — § 7.25.7 on page 317

6.4.14. fo:region-before

Common Usage:

Used in constructing a simple-page-master. This region specifies a viewport/reference pair that is located on the "before" side of the page-reference-area. In lr-tb writing-mode, this region corresponds to the header region. The *overflow* trait controls how much of the underlying region-reference-area is visible; that is, whether the region-reference-area is clipped by its parent region-viewport-area.

Areas:

The fo:region-before formatting object is used to generate one region-viewport-area and one region-reference-area.

In version 1.0 of this recommendation, the values of the *padding* and *border-width* traits must be "0".

The before-edge of the content-rectangle of this region-viewport-area is positioned coincident with the before-edge of the content-rectangle of the page-reference-area generated using the parent fo:simple-page-master. The block-progression-dimension of the region-viewport-area is determined by the *extent* trait on the fo:region-before formatting object.

The inline-progression-dimension of the region-viewport-area is determined by the *precedence* trait on the fo:region-before. If the value of the *precedence* trait is `true`, then the inline-progression-dimension extends up to the start- and after-edges of the content-rectangle of the page-reference-area. In this case, the region-before region-viewport-area acts like a float into areas generated by the region-start and region-end. If the value of the *precedence* trait on the fo:region-before is `false`, then these adjacent regions float into the area generated by the fo:region-before and the extent of the fo:region-before is (effectively) reduced by the incursions of the adjacent regions.

The region-reference-area lies on a canvas underneath the region-viewport-area. The *reference-orientation* trait is used to orient the coordinate system of the region-reference-area relative to the page-reference-area.

The size of the region-reference-area depends on the setting of the *overflow* trait on the region. If the value of that trait is "auto", "hidden", "error-if-overflow", "paginate", or "visible" then the size of the reference-area is the same as the size of the viewport. If the value of the *overflow* trait is "scroll", the size of the reference-area is equal to the size of the viewport in the inline-progression-direction in the

writing-mode for the region and has no constraint in the block-progression-direction (which implies that it grows to hold the distribution of all the content bound to the region).

Trait Derivation:

The *reference-orientation* of the region-viewport-area is taken from the value of the *reference-orientation* trait on the region-master which specifies the region. *reference-orientation* of the region-reference-area is set to "0" and is, therefore, the same as the orientation established by the region-viewport-area.

The remaining traits on the region-viewport-area and region-reference-area are set according to normal rules for determining the values of traits.

Constraints:

The constraints on the size and position of the region-reference-area generated using the fo:region-before are covered in the "Constraints applicable to regions" section of § 6.4.12 – fo:simple-page-master on page 82.

Contents:

EMPTY

The following properties apply to this formatting object:

- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- "clip" — § 7.18.1 on page 268
- "display-align" — § 7.11.4 on page 226
- "extent" — § 7.23.4 on page 289
- "overflow" — § 7.18.2 on page 269
- "precedence" — § 7.23.14 on page 296
- "region-name" — § 7.23.15 on page 297
- "reference-orientation" — § 7.18.3 on page 270
- "writing-mode" — § 7.25.7 on page 317

6.4.15. fo:region-after

Common Usage:

Used in constructing a simple-page-master. This region specifies a viewport/reference pair that is located on the "after" side of the page-reference-area. In lr-tb writing-mode, this region corresponds to the footer region. The *overflow* trait controls how much of the underlying region-reference-area is visible; that is, whether the region-reference-area is clipped by its parent region-viewport-area.

Areas:

The fo:region-after formatting object is used to generate one region-viewport-area and one region-reference-area.

In version 1.0 of this recommendation, the values of the *padding* and *border-width* traits must be "0".

The after-edge of the content-rectangle of this region-viewport-area is positioned coincident with the after-edge of the content-rectangle of the page-reference-area generated using the parent fo:simple-page-master. The block-progression-dimension of the region-viewport-area is determined by the *extent* trait on the fo:region-after formatting object.

The inline-progression-dimension of the region-viewport-area is determined by the *precedence* trait on

the `fo:region-after`. If the value of the *precedence* trait is `true`, then the inline-progression-dimension extends up to the start- and after-edges of the content-rectangle of the page-reference-area. In this case, the region-after region-viewport-area acts like a float into areas generated by the region-start and region-end. If the value of the *precedence* trait on the `fo:region-after` is `false`, then these adjacent regions float into the area generated by the `fo:region-after` and the extent of the `fo:region-after` is (effectively) reduced by the incursions of the adjacent regions.

The region-reference-area lies on a canvas underneath the region-viewport-area. The *reference-orientation* trait is used to orient the coordinate system of the region-reference-area relative to the page-reference-area.

The size of the region-reference-area depends on the setting of the *overflow* trait on the region. If the value of that trait is "auto", "hidden", "error-if-overflow", "paginate", or "visible" then the size of the reference-area is the same as the size of the viewport. If the value of the *overflow* trait is "scroll", the size of the reference-area is equal to the size of the viewport in the inline-progression-direction in the *writing-mode* for the region and has no constraint in block-progression-direction (which implies that it grows to hold the distribution of all the content bound to the region).

Trait Derivation:

The *reference-orientation* of the region-viewport-area is taken from the value of the *reference-orientation* trait on the region-master which specifies the region. *reference-orientation* of the region-reference-area is set to "0" and is, therefore, the same as the orientation established by the region-viewport-area.

The remaining traits on the region-viewport-area and region-reference-area are set according to normal rules for determining the values of traits.

Constraints:

The constraints on the size and position of the region-reference-area generated using the `fo:region-after` are covered in the "Constraints applicable to regions" section of § 6.4.12 – `fo:simple-page-master` on page 82.

Contents:

EMPTY

The following properties apply to this formatting object:

- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- “clip” — § 7.18.1 on page 268
- “display-align” — § 7.11.4 on page 226
- “extent” — § 7.23.4 on page 289
- “overflow” — § 7.18.2 on page 269
- “precedence” — § 7.23.14 on page 296
- “region-name” — § 7.23.15 on page 297
- “reference-orientation” — § 7.18.3 on page 270
- “writing-mode” — § 7.25.7 on page 317

6.4.16. `fo:region-start`

Common Usage:

Used in constructing a simple-page-master. This region specifies a viewport/reference pair that is located on the "start" side of the page-reference-area. In lr-tb writing-mode, this region corresponds to a

left sidebar. The *overflow* trait controls how much of the underlying region-reference-area is visible; that is, whether the region-reference-area is clipped by its parent region-viewport-area.

Areas:

The fo:region-start formatting object is used to generate one region-viewport-area and one region-reference-area.

In version 1.0 of this recommendation, the values of the *padding* and *border-width* traits must be "0".

The start-edge of the content-rectangle of this region-viewport-area is positioned coincident with the start-edge of the content-rectangle of the page-reference-area generated using the parent fo:simple-page-master. The inline-progression-dimension of the region-viewport-area is determined by the *extent* trait on the fo:region-after formatting object.

The block-progression-dimension of the region-viewport-area is determined by the *precedence* trait on the adjacent fo:region-before and the fo:region-after. If the value of the *precedence* trait of the fo:region-before (or, respectively, fo:region-after) is *false*, then the block-progression-dimension extends up to the before- (or, respectively, after-) edge of the content-rectangle of the page-reference-area. In this case, the region-start acts like a float into areas generated by the region-before (respectively, the region-after). If the value of the *precedence* trait on the adjacent regions is *true*, then these adjacent regions float into the area generated by the fo:region-start and the extent of the fo:region-start is (effectively) reduced by the incursions of the adjacent regions with the value of the *precedence* trait equal to *true*.

The region-reference-area lies on a canvas underneath the region-viewport-area. The *reference-orientation* trait is used to orient the coordinate system of the region-reference-area relative to the page-reference-area.

The size of the region-reference-area depends on the setting of the *overflow* trait on the region. If the value of that trait is "auto", "hidden", "error-if-overflow", "paginate", or "visible" then the size of the reference-area is the same as the size of the viewport. If the value of the *overflow* trait is "scroll", the size of the reference-area is equal to the size of the viewport in the inline-progression-direction in the *writing-mode* for the region and has no constraint in block-progression-direction (which implies that it grows to hold the distribution of all the content bound to the region).

Trait Derivation:

The *reference-orientation* of the region-viewport-area is taken from the value of the *reference-orientation* trait on the region-master which specifies the region. *reference-orientation* of the region-reference-area is set to "0" and is, therefore, the same as the orientation established by the region-viewport-area.

The remaining traits on the region-viewport-area and region-reference-area are set according to normal rules for determining the values of traits.

Constraints:

The constraints on the size and position of the region-reference-area generated using the fo:region-start are covered in the "Constraints applicable to regions" section of § 6.4.12 – fo:simple-page-master on page 82.

Contents:

EMPTY

The following properties apply to this formatting object:

- “clip” — § 7.18.1 on page 268
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- “display-align” — § 7.11.4 on page 226
- “extent” — § 7.23.4 on page 289
- “overflow” — § 7.18.2 on page 269
- “region-name” — § 7.23.15 on page 297
- “reference-orientation” — § 7.18.3 on page 270
- “writing-mode” — § 7.25.7 on page 317

6.4.17. fo:region-end

Common Usage:

Used in constructing a simple-page-master. This region specifies a viewport/reference pair that is located on the "end" side of the page-reference-area. In lr-tb writing-mode, this region corresponds to a right sidebar. The *overflow* trait controls how much of the underlying region-reference-area is visible; that is, whether the region-reference-area is clipped by its parent region-viewport-area.

Areas:

The fo:region-end formatting object is used to generate one region-viewport-area and one region-reference-area.

In version 1.0 of this recommendation, the values of the *padding* and *border-width* traits must be "0".

The end-edge of the content-rectangle of this region-viewport-area is positioned coincident with the end-edge of the content-rectangle of the page-reference-area generated using the parent fo:simple-page-master. The inline-progression-dimension of the region-viewport-area is determined by the *extent* trait on the fo:region-after formatting object.

The block-progression-dimension of the region-viewport-area is determined by the *precedence* trait on the adjacent fo:region-before and the fo:region-after. If the value of the *precedence* trait of the fo:region-before (or, respectively, fo:region-after) is `false`, then the block-progression-dimension extends up to the before- (or, respectively, after-) edge of the content-rectangle of the page-reference-area. In this case, the region-end acts like a float into areas generated by the region-before (respectively, the region-after). If the value of the *precedence* trait on the adjacent regions is `true`, then these adjacent regions float into the area generated by the fo:region-end and the extent of the fo:region-end is (effectively) reduced by the incursions of the adjacent regions with the value of the *precedence* trait equal to `true`.

The region-reference-area lies on a canvas underneath the region-viewport-area. The *reference-orientation* trait is used to orient the coordinate system of the region-reference-area relative to the page-reference-area.

The size of the region-reference-area depends on the setting of the *overflow* trait on the region. If the value of that trait is "auto", "hidden", "error-if-overflow", "paginate", or "visible" then the size of the reference-area is the same as the size of the viewport. If the value of the *overflow* trait is "scroll", the size of the reference-area is equal to the size of the viewport in the inline-progression-direction in the *writing-mode* for the region and has no constraint in block-progression-direction (which implies that it grows to hold the distribution of all the content bound to the region).

Trait Derivation:

The *reference-orientation* of the region-viewport-area is taken from the value of the *reference-orientation* trait on the region-master which specifies the region. *reference-orientation* of the region-reference-area is set to "0" and is, therefore, the same as the orientation established by the region-viewport-area.

The remaining traits on the region-viewport-area and region-reference-area are set according to normal rules for determining the values of traits.

Constraints:

The constraints on the size and position of the region-reference-area generated using the fo:region-end are covered in the "Constraints applicable to regions" section of § 6.4.12 – fo:simple-page-master on page 82.

Contents:

EMPTY

The following properties apply to this formatting object:

- “clip” — § 7.18.1 on page 268
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- “display-align” — § 7.11.4 on page 226
- “extent” — § 7.23.4 on page 289
- “overflow” — § 7.18.2 on page 269
- “region-name” — § 7.23.15 on page 297
- “reference-orientation” — § 7.18.3 on page 270
- “writing-mode” — § 7.25.7 on page 317

6.4.18. fo:flow*Common Usage:*

The content of the fo:flow formatting object is a sequence of flow objects that provides the flowing text content that is distributed into pages.

Areas:

The fo:flow formatting object does not generate any areas. The fo:flow formatting object returns a sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:flow. The order of concatenation is the same order as the children are ordered under the fo:flow.

Constraints:

The (implicit) flow-map determines the assignment of the content of the fo:flow to a region.

Contents:

([%block;](#)) +

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- “flow-name” — § 7.23.5 on page 290

6.4.19. fo:static-content

Common Usage:

The fo:static-content formatting object holds a sequence or a tree of formatting objects that is to be presented in a single region or repeated in like-named regions on one or more pages in the page-sequence. Its common use is for repeating or running headers and footers.

This content is repeated, in its entirety, on every page to which it is assigned.

Areas:

The fo:static-content formatting object does not generate any areas. The fo:static-content formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:static-content. The order of concatenation is the same order as the children are ordered under the fo:static-content.

Constraints:

The (implicit) flow-map determines the assignment of the content of the fo:static-content to a region.

The fo:static-content may be processed multiple times and thus the default ordering constraint of section § 4.7.1 – [General Ordering Constraints](#) on page 32 does not apply to the fo:static-content. Instead, it must satisfy the constraint on a per-page basis. Specifically, if *P* is a page-reference-area *C* is an area-class, and *S* is the set of all descendants of *P* of area-class *C* returned to the fo:static-content descendant, then *S* must be properly-ordered.

Contents:

(%block;)+

The following properties apply to this formatting object:

- “flow-name” — § 7.23.5 on page 290

6.4.20. fo:title

Common Usage:

The fo:title formatting object is used to associate a title with a given document. This title may be used by an interactive User Agent to identify the document. For example, the content of the fo:title can be formatted and displayed in a "title" window or in a "tool tip".

Areas:

This formatting object returns the sequence of areas returned by the flow children of this formatting object.

Constraints:

The sequence of returned areas must be the concatenation of the sub-sequences of areas returned by each of the flow children of the fo:title formatting object in the order in which the children occur.

Contents:

(#PCDATA|%inline;)*

An fo:title is not permitted to have an fo:float, fo:footnote or fo:marker as a descendant.

Additionally, an fo:title is not permitted to have as a descendant an fo:block-container that generates an absolutely positioned area.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Font Properties](#) — § 7.7 on page 190
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “baseline-shift” — § 7.11.3 on page 225
- “color” — § 7.15.1 on page 258
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “visibility” — § 7.26.8 on page 322
- “z-index” — § 7.26.9 on page 323

6.5. Block-level Formatting Objects

6.5.1. Introduction

The fo:block formatting object is used for formatting paragraphs, titles, figure captions, table titles, etc. The following example illustrates the usage of the fo:block in a style sheet.

6.5.1.1. Example

6.5.1.1.1. Chapter and Section Titles, Paragraphs

Input sample:

```
<doc>
  <chapter>
    <title>Chapter title</title>
    <section>
      <title>First section title</title>
      <paragraph>Section one's first paragraph.</paragraph>
      <paragraph>Section one's second paragraph.</paragraph>
    </section>
    <section>
      <title>Second section title</title>
      <paragraph>Section two's only paragraph.</paragraph>
    </section>
  </chapter>
</doc>
```

In this example the chapter title appears at the top of the page (its “space-before” is discarded).

Space between chapter title and first section title is (8pt,8pt,8pt): the chapter title's “space-after” has a higher precedence than the section title's “space-before” (which takes on the initial value of zero), so the latter is discarded.

Space between the first section title and section one's first paragraph is (6pt,6pt,6pt): the section title's “space-after” has higher precedence than the paragraph's “space-before”, so the latter is discarded.

Space between the two paragraphs is (6pt,8pt,10pt): the “space-after” the first paragraph is discarded because its precedence is equal to that of the “space-before” the next paragraph, and the optimum of the “space-after” of the first paragraph is greater than the optimum of the “space-before” of the second

paragraph.

Space between the second paragraph of the first section and the title of the second section is (12pt,12pt,12pt): the "space-after" the paragraph is discarded because its precedence is equal to that of the "space-before" of the section title, and the optimum of the "space-after" of the paragraph is less than the optimum of the "space-before" of the section title.

The indent on the first line of the first paragraph in section one and the only paragraph in section two is zero; the indent on the first line of the second paragraph in section one is 2pc.

XSL Stylesheet:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="chapter">
  <fo:block break-before="page">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="chapter/title">
  <fo:block text-align="center" space-after="8pt"
    space-before="16pt" space-after.precedence="3">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="section">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="section/title">
  <fo:block text-align="center" space-after="6pt"
    space-before="12pt" space-before.precedence="0"
    space-after.precedence="3">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="paragraph[1]" priority="1">
  <fo:block text-indent="0pc" space-after="7pt"
    space-before.minimum="6pt" space-before.optimum="8pt"
    space-before.maximum="10pt">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="paragraph">
  <fo:block text-indent="2pc" space-after="7pt"
    space-before.minimum="6pt" space-before.optimum="8pt"
    space-before.maximum="10pt">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
</xsl:stylesheet>
```

Result Instance: elements and attributes in the fo: namespace

```

<fo:block break-before="page">
  <fo:block text-align="center" space-after="8pt"
    space-before="16pt"
    space-after.precedence="3">Chapter title
  </fo:block>
  <fo:block text-align="center" space-after="6pt"
    space-before="12pt" space-before.precedence="0"
    space-after.precedence="3">First section title
  </fo:block>
  <fo:block text-indent="0pc" space-after="7pt"
    space-before.minimum="6pt" space-before.optimum="8pt"
    space-before.maximum="10pt">Section one's first paragraph.
  </fo:block>
  <fo:block text-indent="2pc" space-after="7pt"
    space-before.minimum="6pt" space-before.optimum="8pt"
    space-before.maximum="10pt">Section one's second paragraph.
  </fo:block>
  <fo:block text-align="center" space-after="6pt"
    space-before="12pt" space-before.precedence="0"
    space-after.precedence="3">Second section title
  </fo:block>
  <fo:block text-indent="0pc" space-after="7pt"
    space-before.minimum="6pt" space-before.optimum="8pt"
    space-before.maximum="10pt">Section two's only paragraph.
  </fo:block>
</fo:block>

```

6.5.2. fo:block

Common Usage:

The fo:block formatting object is commonly used for formatting paragraphs, titles, headlines, figure and table captions, etc.

Areas:

The fo:block formatting object generates one or more *normal* block-areas. The fo:block returns these areas, any *page-level-out-of-line* areas, and any *reference-level-out-of-line* areas returned by the children of the fo:block.

Trait Derivation:

The *half-leading* trait is set to 1/2 the difference of the computed value of the *line-height* property and the computed value of the *font-size* property.

Constraints:

No area may have more than one normal child area returned by the same fo:block formatting object.

The children of each normal area returned by an fo:block must satisfy the constraints specified in § 4.7.2 – [Line-building](#) on page 33.

In addition the constraints imposed by the traits derived from the properties applicable to this formatting object must be satisfied. The geometric constraints are rigorously defined in § 4 – [Area Model](#) on page 13.

Contents:

```
(#PCDATA|%inline;|%block;)*
```

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children, optionally followed by an fo:initial-property-set.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Font Properties](#) — § 7.7 on page 190
- [Common Hyphenation Properties](#) — § 7.8 on page 202
- [Common Margin Properties-Block](#) — § 7.9 on page 206
- “break-after” — § 7.17.1 on page 264
- “break-before” — § 7.17.2 on page 265
- “color” — § 7.15.1 on page 258
- “font-height-override-after” — § 7.25.2 on page 313
- “font-height-override-before” — § 7.25.3 on page 313
- “hyphenation-keep” — § 7.13.1 on page 240
- “hyphenation-ladder-count” — § 7.13.2 on page 240
- “id” — § 7.26.2 on page 320
- “keep-together” — § 7.17.3 on page 265
- “keep-with-next” — § 7.17.4 on page 266
- “keep-with-previous” — § 7.17.5 on page 267
- “last-line-end-indent” — § 7.13.3 on page 241
- “linefeed-treatment” — § 7.13.7 on page 244
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “line-stacking-strategy” — § 7.13.6 on page 243
- “orphans” — § 7.17.6 on page 268
- “relative-position” — § 7.18.4 on page 272
- “space-treatment” — § 7.13.8 on page 245
- “span” — § 7.18.5 on page 272
- “text-align” — § 7.13.9 on page 246
- “text-align-last” — § 7.13.10 on page 247
- “text-indent” — § 7.13.11 on page 248
- “visibility” — § 7.26.8 on page 322
- “white-space-collapse” — § 7.13.12 on page 249
- “widows” — § 7.17.7 on page 268
- “wrap-option” — § 7.13.13 on page 250
- “z-index” — § 7.26.9 on page 323

6.5.3. fo:block-container

Common Usage:

The fo:block-container flow object is used to generate a block-level reference-area, typically containing text blocks with a different writing-mode. In addition, it can also be used with a different reference-orientation to rotate its content.

NOTE: The use of this flow object is not required for changing the inline-progression-direction only; in that case the Unicode bidi algorithm and the fo:bidi-override are sufficient.

Areas:

The fo:block-container formatting object generates one or more *viewport/reference pairs*. The fo:block-container returns these areas and any *page-level-out-of-line* areas returned by the children of the fo:block-container.

Trait Derivation:

The areas generated by the fo:block-container formatting object have a value of "true" for the *is-reference-area*.

The size of the viewport-area and the reference-area has to be fixed in the inline-progression-direction. It must be specified unless the inline-progression-direction is parallel to the inline-progression-direction of the reference-area into which the areas generated by this flow object are placed.

Constraints:

The children of each reference-area generated by an fo:block-container formatting object must be normal *block-areas* returned by the children of the fo:block-container, must be *properly stacked*, and must be *properly ordered*.

Any *reference-level-out-of-line* areas returned by the children of the fo:block-container are handled as described in § 6.10.2 – fo:float on page 152.

Contents:

(%block;)+

In addition an fo:block-container that does not generate an *absolutely positioned* area may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Absolute Position Properties](#) — § 7.4 on page 163
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Block](#) — § 7.9 on page 206
- [“block-progression-dimension”](#) — § 7.12.1 on page 230
- [“break-after”](#) — § 7.17.1 on page 264
- [“break-before”](#) — § 7.17.2 on page 265
- [“clip”](#) — § 7.18.1 on page 268
- [“display-align”](#) — § 7.11.4 on page 226
- [“height”](#) — § 7.12.4 on page 233
- [“id”](#) — § 7.26.2 on page 320
- [“inline-progression-dimension”](#) — § 7.12.5 on page 234
- [“keep-together”](#) — § 7.17.3 on page 265
- [“keep-with-next”](#) — § 7.17.4 on page 266
- [“keep-with-previous”](#) — § 7.17.5 on page 267
- [“overflow”](#) — § 7.18.2 on page 269
- [“reference-orientation”](#) — § 7.18.3 on page 270
- [“span”](#) — § 7.18.5 on page 272
- [“width”](#) — § 7.12.12 on page 239
- [“writing-mode”](#) — § 7.25.7 on page 317

6.6. Inline-level Formatting Objects

6.6.1. Introduction

Inline-level formatting objects are most commonly used to format a portion of text or for generating rules and leaders. There are many other uses. The following examples illustrate some of these uses of inline-level formatting objects.

- putting the first line of a paragraph into small-caps,
- turning a normally inline formatting object, `fo:external-graphic`, into a block by "wrapping" with an `fo:block` formatting object,
- formatting a running footer containing the word "Page" followed by a page number.

6.6.1.1. Examples

6.6.1.1.1. First Line of Paragraph in Small-caps

Input sample:

```
<doc>
<p>This is the text of a paragraph that is going to be
presented with the first line in small-caps.</p>
</doc>
```

XSL Stylesheet:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>
<xsl:template match="p">
  <fo:block>
    <fo:initial-property-set font-variant="small-caps"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
</xsl:stylesheet>
```

Result instance: elements and attributes in the `fo:` namespace

```
<fo:block>
  <fo:initial-property-set font-variant="small-caps">
  </fo:initial-property-set>This is the text of a paragraph that is going to be
presented with the first line in small-caps.
</fo:block>
```

6.6.1.1.2. Figure with a Photograph

Input sample:

```
<doc>
  <figure>
    <photo image="TH0317A.jpg"/>
    <caption>C'ien Tamlung of C'ien Mai</caption>
  </figure>
```

```
</doc>
```

In this example the image (an `fo:external-graphic`) is placed as a centered block-level object. The caption is centered with 10mm indents.

XSL Stylesheet:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>
<xsl:template match="figure">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="photo">
  <fo:block text-align="center">
    <fo:external-graphic src="{@image}"/>
  </fo:block>
</xsl:template>
<xsl:template match="caption">
  <fo:block space-before="3pt" text-align="center"
    start-indent="10mm" end-indent="10mm">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
</xsl:stylesheet>
```

fo: element and attribute tree:

```
<fo:block>
  <fo:block text-align="center">
    <fo:external-graphic src="TH0317A.jpg"/>
  </fo:block>
  <fo:block space-before="3pt" text-align="center" start-indent="10mm"
    end-indent="10mm">C'ien Tamlung of C'ien Mai</fo:block>
</fo:block>
```

6.6.1.1.3. Page numbering and page number reference

Input sample:

```
<!DOCTYPE doc SYSTEM "pgref.dtd">
<doc>
  <chapter id="x"><title>Chapter</title>
    <p>Text</p>
  </chapter>
  <chapter><title>Chapter</title>
    <p>For a description of X see <ref refid="x"/>.</p>
  </chapter>
</doc>
```

In this example each page has a running footer containing the word "Page" followed by the page number. The "ref" element generates the word "page" followed by the page number of the page on which the referenced by the "refid" attribute was placed.

XSL Stylesheet:

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>
<xsl:template match="doc">
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master master-name="page"
        page-height="297mm" page-width="210mm"
        margin-top="20mm" margin-bottom="10mm"
        margin-left="25mm" margin-right="25mm">
        <fo:region-body
          margin-top="0mm" margin-bottom="15mm"
          margin-left="0mm" margin-right="0mm"/>
        <fo:region-after extent="10mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-name="page">
      <fo:static-content flow-name="xsl-region-after">
        <fo:block>
          <xsl:text>Page </xsl:text>
          <fo:page-number/>
        </fo:block>
      </fo:static-content>
      <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates/>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>
<xsl:template match="chapter/title">
  <fo:block id="{generate-id(.)}">
    <xsl:number level="multiple" count="chapter" format="1. "/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="p">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="ref">
  <xsl:text>page </xsl:text>
  <fo:page-number-citation refid="{generate-id(id(@refid)/title)}"/>
</xsl:template>
</xsl:stylesheet>

```

Result Instance: elements and attributes in the fo: namespace

```

<fo:root>
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page"

```

```

    page-height="297mm" page-width="210mm"
    margin-top="20mm" margin-bottom="10mm"
    margin-left="25mm" margin-right="25mm">
    <fo:region-body margin-top="0mm" margin-bottom="15mm"
      margin-left="0mm" margin-right="0mm"/>
    <fo:region-after extent="10mm"/>
  </fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence master-name="page">
  <fo:static-content flow-name="xsl-region-after">
    <fo:block>Page <fo:page-number/>
  </fo:block>
</fo:static-content>
  <fo:flow flow-name="xsl-region-body">
    <fo:block id="N5">1. Chapter</fo:block>
    <fo:block>Text</fo:block>
    <fo:block id="N13">2. Chapter</fo:block>
    <fo:block>For a description of X see page <fo:page-number-citation
refid="N5"/>
  </fo:block>
</fo:flow>
</fo:page-sequence>
</fo:root>

```

6.6.2. fo:bidi-override

Common Usage:

The fo:bidi-override formatting object is used when the Unicode-bidi algorithm fails. It forces a string of text to be written in a specific direction.

Areas:

The fo:bidi-override formatting object generates one or more *normal inline-areas*. The fo:bidi-override returns these areas, any *page-level-out-of-line* areas, and any *reference-level-out-of-line* areas returned by the children of the fo:bidi-override.

Trait Derivation:

The direction traits are derived from the "writing-mode", "direction", and "unicode-bidi" properties as described in § 5.5.3 – [Writing-mode and Direction Properties](#) on page 47.

Constraints:

No area may have more than one normal child area returned by the same fo:bidi-override formatting object.

The children of each normal area returned by an fo:bidi-override must satisfy the constraints specified in § 4.7.3 – [Inline-building](#) on page 34.

Contents:

```
(#PCDATA|%inline;|%block;)*
```

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Font Properties](#) — § 7.7 on page 190
- “color” — § 7.15.1 on page 258
- “direction” — § 7.25.1 on page 312
- “id” — § 7.26.2 on page 320
- “letter-spacing” — § 7.14.2 on page 251
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “relative-position” — § 7.18.4 on page 272
- “score-spaces” — § 7.26.6 on page 321
- “text-shadow” — § 7.14.5 on page 254
- “text-transform” — § 7.14.6 on page 255
- “unicode-bidi” — § 7.25.6 on page 316
- “word-spacing” — § 7.14.8 on page 256

6.6.3. fo:character

Common Usage:

The fo:character flow object represents a character that is mapped to a glyph for presentation. It is an atomic unit to the formatter.

When the result tree is interpreted as a tree of formatting objects, a character in the result tree is treated as if it were an empty element of type fo:character with a character attribute equal to the Unicode representation of the character. The semantics of an "auto" value for character properties, which is typically their initial value, are based on the Unicode codepoint. Overrides may be specified in an implementation-specific manner.

NOTE: In a stylesheet the explicit creation of an fo:character may be used to explicitly override the default mapping.

Unicode Special-use tag characters on Plane 14 need not be supported.

NOTE: Unicode TR20 ([[UNICODE TR20](#)]), in fact, plans to declare very clearly that they are not suitable together with markup.

Areas:

The fo:character formatting object generates and returns a single *normal inline-area*.

Constraints:

The dimensions of the area is determined by the font metrics for the glyph.

When formatting an fo:character with a "treat-as-wordspace" value of "yes", the User Agent may use a different method for determining the *inline-progression-dimension* of the area.

NOTE: Such methods typically make use of a word-space value stored in the font, or a formatter defined word-space value.

Contents:

EMPTY

The following properties apply to this formatting object:

- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Font Properties](#) — § 7.7 on page 190
- [Common Hyphenation Properties](#) — § 7.8 on page 202
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “alignment-adjust” — § 7.11.1 on page 221
- “treat-as-word-space” — § 7.14.7 on page 256
- “alignment-baseline” — § 7.11.2 on page 223
- “baseline-shift” — § 7.11.3 on page 225
- “character” — § 7.14.1 on page 250
- “color” — § 7.15.1 on page 258
- “dominant-baseline” — § 7.11.5 on page 227
- “font-height-override-after” — § 7.25.2 on page 313
- “font-height-override-before” — § 7.25.3 on page 313
- “glyph-orientation-horizontal” — § 7.25.4 on page 314
- “glyph-orientation-vertical” — § 7.25.5 on page 315
- “id” — § 7.26.2 on page 320
- “keep-with-next” — § 7.17.4 on page 266
- “keep-with-previous” — § 7.17.5 on page 267
- “letter-spacing” — § 7.14.2 on page 251
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “relative-position” — § 7.18.4 on page 272
- “score-spaces” — § 7.26.6 on page 321
- “suppress-at-line-break” — § 7.14.3 on page 252
- “text-decoration” — § 7.14.4 on page 253
- “text-shadow” — § 7.14.5 on page 254
- “text-transform” — § 7.14.6 on page 255
- “word-spacing” — § 7.14.8 on page 256

6.6.4. fo:initial-property-set

Common Usage:

The fo:initial-property-set auxiliary formatting object specifies formatting properties for the first line of an fo:block.

NOTE: It is analogous to the CSS first-line pseudo-element.

In future versions of this recommendation a property controlling the number of lines, or the "depth" that these initial properties apply to may be added.

Areas:

The fo:initial-property-set formatting object does not generate or return any areas. It simply holds a set of traits that are applicable to the first line-area of the area that has a value of "true" for the *is-first* trait and that was generated by the parent fo:block of the fo:initial-property-set.

Trait Derivation:

The traits on the fo:initial-property-set are taken into account as traits constraining the first line as if the

child inline formatting objects of the fo:block, or parts of them in the case of a line-break, that were used in formatting the first line were enclosed by an fo:wrapper, as a direct child of the fo:block, with those traits.

Constraints:

None.

Contents:

EMPTY

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Font Properties](#) — § 7.7 on page 190
- “color” — § 7.15.1 on page 258
- “id” — § 7.26.2 on page 320
- “letter-spacing” — § 7.14.2 on page 251
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “relative-position” — § 7.18.4 on page 272
- “score-spaces” — § 7.26.6 on page 321
- “text-decoration” — § 7.14.4 on page 253
- “text-shadow” — § 7.14.5 on page 254
- “text-transform” — § 7.14.6 on page 255
- “word-spacing” — § 7.14.8 on page 256

6.6.5. fo:external-graphic

Common Usage:

The fo:external-graphic flow object is used for a graphic where the graphics data resides outside of the fo:element tree.

Areas:

The fo:external-graphic formatting object generates and returns one inline-level viewport-area and one reference-area containing the external graphic.

NOTE: An fo:external-graphic may be block-level by enclosing it in an fo:block.

Constraints:

The viewport's size is determined by the *height* and *width* traits. For values of "auto", the content size of the graphic is used.

The content size of a graphic is determined by taking the intrinsic size of the graphic and scaling as specified by the *content-height*, *content-width*, and *scaling* traits. If one of the content-height or content-width is not "auto", the same scale factor (as calculated from the specified non-auto value) is applied equally to both directions.

Once scaled, the reference-area is aligned with respect to the viewport-area using the *text-align* and *display-align* traits. If it is too large for the viewport-area, the graphic is aligned as if it would fit and the

overflow trait controls the clipping, scroll bars, etc.

In the case when the graphics format does not specify an intrinsic size of the graphic the size is determined in an implementation-defined manner.

NOTE: For example, using a size of 1/96" as the size of one pixel for rasterized images.

Contents:

EMPTY

The following properties apply to this formatting object:

- “display-align” — § 7.11.4 on page 226
- “text-align” — § 7.13.9 on page 246
- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “alignment-adjust” — § 7.11.1 on page 221
- “alignment-baseline” — § 7.11.2 on page 223
- “baseline-shift” — § 7.11.3 on page 225
- “block-progression-dimension” — § 7.12.1 on page 230
- “content-height” — § 7.12.2 on page 231
- “content-type” — § 7.26.1 on page 319
- “content-width” — § 7.12.3 on page 232
- “display-align” — § 7.11.4 on page 226
- “dominant-baseline” — § 7.11.5 on page 227
- “height” — § 7.12.4 on page 233
- “id” — § 7.26.2 on page 320
- “inline-progression-dimension” — § 7.12.5 on page 234
- “keep-with-next” — § 7.17.4 on page 266
- “keep-with-previous” — § 7.17.5 on page 267
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “relative-position” — § 7.18.4 on page 272
- “overflow” — § 7.18.2 on page 269
- “scaling” — § 7.12.10 on page 238
- “scaling-method” — § 7.12.11 on page 238
- “src” — § 7.26.7 on page 322
- “text-align” — § 7.13.9 on page 246
- “width” — § 7.12.12 on page 239

6.6.6. fo:instream-foreign-object

Common Usage:

The fo:instream-foreign-object flow object is used for an inline graphic or other "generic" object where the object data resides as descendants of the fo:instream-foreign-object, typically as an XML element subtree in a non-XSL namespace.

NOTE: A common format is SVG.

Areas:

The fo:instream-foreign-object formatting object generates and returns one inline viewport-area and one reference-area containing the instream-foreign-object.

Constraints:

The viewport's size is determined by the *height* and *width* traits. For values of "auto", the content size of the instream foreign object is used.

The content size of an instream-foreign-object is determined by taking the intrinsic size of the object and scaling as specified by the *content-height*, *content-width*, and *scaling* traits. If one of the content-height or content-width is not "auto", the same scale factor (as calculated from the specified non-auto value) is applied equally to both directions.

Once scaled, the reference-area is aligned with respect to the viewport-area using the *text-align* and *display-align* traits. If it is too large for the viewport-area, the instream-foreign-object is aligned as if it would fit and the *overflow* trait controls the clipping, scroll bars, etc.

In the case when the instream-foreign-object does not specify an intrinsic size of the object, the size is determined in an implementation defined manner.

Contents:

The fo:instream-foreign-object flow object has a child from a non-XSL namespace. The permitted structure of this child is that defined for that namespace.

The fo:instream-foreign-object flow object may have additional attributes in the non-XSL namespace. These, as well as the xsl defined properties, are made available to the processor of the content of the flow object. Their semantics is defined by that namespace.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “[alignment-adjust](#)” — § 7.11.1 on page 221
- “[alignment-baseline](#)” — § 7.11.2 on page 223
- “[baseline-shift](#)” — § 7.11.3 on page 225
- “[block-progression-dimension](#)” — § 7.12.1 on page 230
- “[content-height](#)” — § 7.12.2 on page 231
- “[content-type](#)” — § 7.26.1 on page 319
- “[content-width](#)” — § 7.12.3 on page 232
- “[display-align](#)” — § 7.11.4 on page 226
- “[dominant-baseline](#)” — § 7.11.5 on page 227
- “[height](#)” — § 7.12.4 on page 233
- “[id](#)” — § 7.26.2 on page 320
- “[inline-progression-dimension](#)” — § 7.12.5 on page 234
- “[keep-with-next](#)” — § 7.17.4 on page 266
- “[keep-with-previous](#)” — § 7.17.5 on page 267
- “[line-height](#)” — § 7.13.4 on page 241
- “[line-height-shift-adjustment](#)” — § 7.13.5 on page 243
- “[overflow](#)” — § 7.18.2 on page 269
- “[relative-position](#)” — § 7.18.4 on page 272

- “[scaling](#)” — § 7.12.10 on page 238
- “[scaling-method](#)” — § 7.12.11 on page 238
- “[text-align](#)” — § 7.13.9 on page 246
- “[width](#)” — § 7.12.12 on page 239

6.6.7. [fo:inline](#)

Common Usage:

The [fo:inline](#) formatting object is commonly used for formatting a portion of text with a background or enclosing it in a border.

Areas:

The [fo:inline](#) formatting object generates one or more *normal inline-areas*. The [fo:inline](#) returns these areas, any *page-level-out-of-line* areas, and any *reference-level-out-of-line* areas returned by the children of the [fo:inline](#).

Constraints:

No area may have more than one normal child area returned by the same [fo:inline](#) formatting object.

The children of each normal area returned by an [fo:inline](#) must satisfy the constraints specified in § 4.7.3 – [Inline-building](#) on page 34.

In addition the constraints imposed by the traits derived from the properties applicable to this formatting object must be satisfied. The geometric constraints are rigorously defined in § 4 – [Area Model](#) on page 13.

Contents:

```
(#PCDATA|%inline;|%block;)*
```

In addition this formatting object may have a sequence of zero or more [fo:markers](#) as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Font Properties](#) — § 7.7 on page 190
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “[alignment-adjust](#)” — § 7.11.1 on page 221
- “[alignment-baseline](#)” — § 7.11.2 on page 223
- “[baseline-shift](#)” — § 7.11.3 on page 225
- “[color](#)” — § 7.15.1 on page 258
- “[dominant-baseline](#)” — § 7.11.5 on page 227
- “[id](#)” — § 7.26.2 on page 320
- “[keep-together](#)” — § 7.17.3 on page 265
- “[keep-with-next](#)” — § 7.17.4 on page 266
- “[keep-with-previous](#)” — § 7.17.5 on page 267
- “[line-height](#)” — § 7.13.4 on page 241
- “[line-height-shift-adjustment](#)” — § 7.13.5 on page 243
- “[relative-position](#)” — § 7.18.4 on page 272
- “[text-decoration](#)” — § 7.14.4 on page 253
- “[visibility](#)” — § 7.26.8 on page 322

-
- “z-index” — § 7.26.9 on page 323

6.6.8. fo:inline-container

Common Usage:

The fo:inline-container flow object is used to generate an inline reference-area, typically containing text blocks with a different writing-mode.

NOTE: The use of this flow object is not required for bi-directional text; in this case the Unicode bidi algorithm and the fo:bidi-override are sufficient.

Areas:

The fo:inline-container formatting object generates one or more *viewport/reference pairs*. The viewport-areas generated by the fo:inline-container are *normal inline-level* areas. The fo:inline-container returns these areas and any *page-level-out-of-line* areas returned by the children of the fo:inline-container.

Trait Derivation:

The areas generated by the fo:inline-container formatting object have a value of "true" for the *is-reference-area*.

The size of the viewport-area and the reference-area has to be fixed in the inline-progression-direction. It must be specified unless the inline-progression-direction is parallel to the inline-progression-direction of the reference-area into which the areas generated by this flow object are placed.

The values in the baseline-table of this object are calculated as follows:

baseline

If the writing mode has a block-progression-direction that is parallel to the block-progression-direction of the parent: the alignment-point is at the position of the dominant-baseline of the first descendant line-area. If there is no such line-area the alignment-point is at the position of the after-edge of the allocation rectangle.

If the writing mode has a block-progression-direction that is not parallel to the block-progression-direction of the parent: the alignment-point is at the position that is half way between the before-edge and after-edge of the content rectangle.

before-edge

The alignment-point is at the position of the before-edge of the allocation rectangle.

text-before-edge

The alignment-point is at the position that is the closest to the before-edge of the allocation rectangle selected from the two candidate edges. If the writing mode has a block-progression-direction that is parallel to the block-progression-direction of the parent the candidate edges are the before-edge and the after-edge of the content rectangle; if it is not, the candidate edges are the start-edge and the end-edge of the content rectangle.

middle

The alignment-point is at the position that is half way between the before-edge and after-edge of the allocation rectangle.

after-edge

The alignment-point is at the position of the after-edge of the allocation rectangle.

text-after-edge

The alignment-point is at the position that is the closest to the after-edge of the allocation rectangle selected from the two candidate edges. If the writing mode has a block-progression-direction that is parallel to the block-progression-direction of the parent the candidate edges are the before-edge and the after-edge of the content rectangle; if it is not, the candidate edges are the start-edge and the end-edge of the content rectangle.

ideographic

The alignment-point is at the position that is 7/10 of the distance from the before-edge of the allocation rectangle to the after-edge of the allocation rectangle.

alphabetic

The alignment-point is at the position that is 6/10 of the distance from the before-edge of the allocation rectangle to the after-edge of the allocation rectangle.

hanging

The alignment-point is at the position that is 2/10 of the distance from the before-edge of the allocation rectangle to the after-edge of the allocation rectangle.

mathematical

The alignment-point is at the position that is 5/10 of the distance from the before-edge of the allocation rectangle to the after-edge of the allocation rectangle.

Constraints:

No area may have more than one normal child area returned by the same fo:inline-container formatting object.

The children of each reference-area generated by an fo:inline-container formatting object must be normal *block-areas* returned by the children of the fo:inline-container, must be *properly stacked*, and must be *properly ordered*.

Any *reference-level-out-of-line* areas returned by the children of the fo:inline-container are handled as described in § 6.10.2 – fo:float on page 152.

Contents:

(%block;)+

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “[alignment-adjust](#)” — § 7.11.1 on page 221
- “[alignment-baseline](#)” — § 7.11.2 on page 223
- “[baseline-shift](#)” — § 7.11.3 on page 225
- “[block-progression-dimension](#)” — § 7.12.1 on page 230

- “clip” — § 7.18.1 on page 268
- “display-align” — § 7.11.4 on page 226
- “dominant-baseline” — § 7.11.5 on page 227
- “height” — § 7.12.4 on page 233
- “id” — § 7.26.2 on page 320
- “inline-progression-dimension” — § 7.12.5 on page 234
- “keep-together” — § 7.17.3 on page 265
- “keep-with-next” — § 7.17.4 on page 266
- “keep-with-previous” — § 7.17.5 on page 267
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “overflow” — § 7.18.2 on page 269
- “reference-orientation” — § 7.18.3 on page 270
- “relative-position” — § 7.18.4 on page 272
- “width” — § 7.12.12 on page 239
- “writing-mode” — § 7.25.7 on page 317

6.6.9. fo:leader

Common Usage:

The fo:leader formatting object is often used:

- in TOCs to generate sequences of "." glyphs that separate titles from page numbers
- to create entry fields in fill-in-the-blank forms
- to create horizontal rules for use as separators

Areas:

The fo:leader formatting object generates and returns a single normal *inline-area*.

Trait Derivation:

If the value of the *leader-pattern* is "use-content" the *block-progression-dimension* of the content-rectangle is determined in the same manner as for line-areas; otherwise it is determined by the *rule-thickness* trait.

Constraints:

If the leader's minimum length is too long to place in the line-area, the leader will begin a new line. If it is too long to be placed in a line by itself, it will overflow the line and potentially overflow the reference-area in accordance with that container's *overflow* trait.

The fo:leader formatting object can have any inline formatting objects and characters as its children, except that fo:leaders may not be nested. Its children are ignored unless the value of the *leader-pattern* trait is "use-content".

NOTE: If the value of the *leader-pattern* trait is "use-content" and the fo:leader has no children, the leader shall be filled with blank space.

The inline-area generated by the fo:leader has a dimension in the inline-progression-direction which shall be at least the *leader-length.minimum* and at most the *leader-length.maximum*.

For lines-areas that have been specified to be justified, the justified line-area must honor the

leader-alignment trait of any inline-areas generated by fo:leaders.

If the value of the *leader-pattern* trait is "dots" or "use-content", the following constraint applies:

The inline-area generated by the fo:leader has as its children the areas returned by children of the fo:leader, or obtained by formatting the pattern specified in the *leader-pattern* trait, repeated an integral number of times. If the width of even a single repetition is larger than the dimension of the inline-area in the inline-progression-direction, the inline-area shall be filled with blank space. The space-start and space-end of the child areas is set to account for the constraints specified in the *leader-pattern-width* and *leader-alignment* traits.

NOTE: If it is desired that the leader should stretch to fill all available space on a line, the maximum length of the leader should be specified to be at least as large as the column width.

NOTE: The alignment of the leader may be script specific and may require indicating what alignment point is required, because it is different from the default alignment for the script. For example, in some usage of Indic scripts the leader is aligned at the alphabetic baseline.

NOTE: An fo:leader can be wrapped in an fo:block to create a rule for separating or decorating block-areas.

Contents:

(#PCDATA|[%inline;](#))*

The content must not contain an fo:leader, fo:inline-container, fo:block-container, fo:float, fo:footnote, or fo:marker either as a direct child or as a descendant.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Font Properties](#) — § 7.7 on page 190
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “alignment-adjust” — § 7.11.1 on page 221
- “alignment-baseline” — § 7.11.2 on page 223
- “baseline-shift” — § 7.11.3 on page 225
- “color” — § 7.15.1 on page 258
- “dominant-baseline” — § 7.11.5 on page 227
- “font-height-override-after” — § 7.25.2 on page 313
- “font-height-override-before” — § 7.25.3 on page 313
- “id” — § 7.26.2 on page 320
- “leader-alignment” — § 7.19.1 on page 273
- “leader-length” — § 7.19.4 on page 275
- “leader-pattern” — § 7.19.2 on page 273
- “leader-pattern-width” — § 7.19.3 on page 274
- “relative-position” — § 7.18.4 on page 272
- “rule-style” — § 7.19.5 on page 275
- “rule-thickness” — § 7.19.6 on page 276
- “letter-spacing” — § 7.14.2 on page 251
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243

- “text-shadow” — § 7.14.5 on page 254
- “visibility” — § 7.26.8 on page 322
- “word-spacing” — § 7.14.8 on page 256
- “z-index” — § 7.26.9 on page 323

6.6.10. fo:page-number

Common Usage:

The fo:page-number formatting object is used to obtain an inline-area whose content is the page-number for the page on which the inline-area is placed.

Areas:

The fo:page-number formatting object generates and returns a single normal *inline-area*.

Constraints:

The child areas of this inline-area are the same as the result of formatting a result-tree fragment consisting of fo:character flow objects; one for each character in the page-number string and with only the "character" property specified.

The page-number string is obtained by converting the page-number for the page on which the inline-area is placed in accordance with the number to string conversion properties specified on the ancestor fo:page-sequence.

Contents:

EMPTY

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Font Properties](#) — § 7.7 on page 190
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “alignment-adjust” — § 7.11.1 on page 221
- “alignment-baseline” — § 7.11.2 on page 223
- “baseline-shift” — § 7.11.3 on page 225
- “dominant-baseline” — § 7.11.5 on page 227
- “id” — § 7.26.2 on page 320
- “keep-with-next” — § 7.17.4 on page 266
- “keep-with-previous” — § 7.17.5 on page 267
- “letter-spacing” — § 7.14.2 on page 251
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “relative-position” — § 7.18.4 on page 272
- “score-spaces” — § 7.26.6 on page 321
- “text-decoration” — § 7.14.4 on page 253
- “text-shadow” — § 7.14.5 on page 254
- “text-transform” — § 7.14.6 on page 255
- “word-spacing” — § 7.14.8 on page 256

6.6.11. fo:page-number-citation

Common Usage:

The fo:page-number-citation is used to reference the page-number for the page containing the first *normal* area returned by the cited formatting object.

NOTE: It may be used to provide the page-numbers in the table of contents, cross-references, and index entries.

Areas:

The fo:page-number-citation formatting object generates and returns a single normal *inline-area*.

Constraints:

The *cited page-number* is the number of the page containing, as a descendant, the first normal area returned by the formatting object with an *id* trait matching the *ref-id* trait of the fo:page-number-citation (the referenced formatting object).

The *cited page-number string* is obtained by converting the cited page-number in accordance with the number to string conversion properties specified on the ancestor fo:page-sequence of the referenced formatting object.

The child areas of the generated inline-area are the same as the result of formatting a result-tree fragment consisting of fo:character flow objects; one for each character in the cited page-number string and with only the "character" property specified.

Contents:

EMPTY

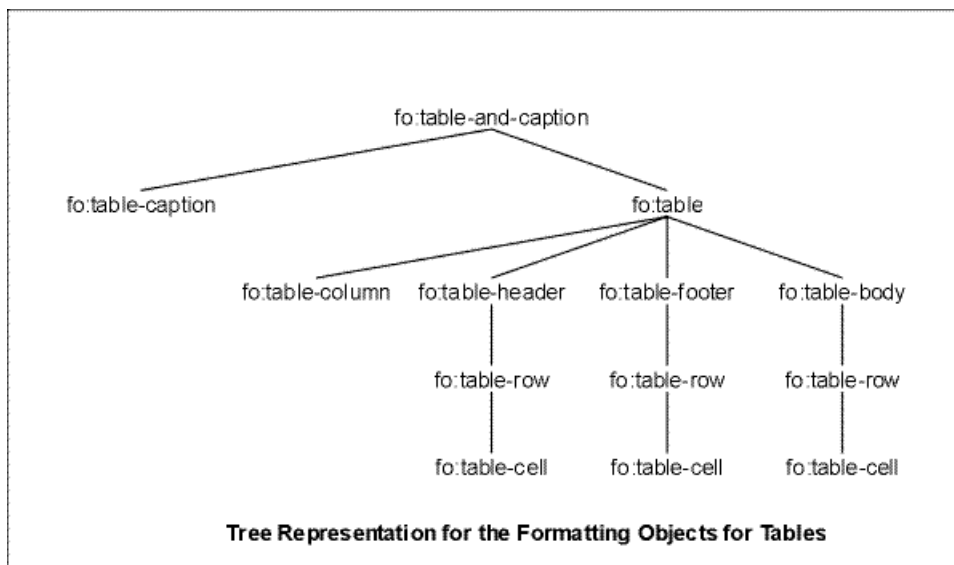
The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Font Properties](#) — § 7.7 on page 190
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- “alignment-adjust” — § 7.11.1 on page 221
- “alignment-baseline” — § 7.11.2 on page 223
- “baseline-shift” — § 7.11.3 on page 225
- “dominant-baseline” — § 7.11.5 on page 227
- “id” — § 7.26.2 on page 320
- “keep-with-next” — § 7.17.4 on page 266
- “keep-with-previous” — § 7.17.5 on page 267
- “letter-spacing” — § 7.14.2 on page 251
- “line-height” — § 7.13.4 on page 241
- “line-height-shift-adjustment” — § 7.13.5 on page 243
- “ref-id” — § 7.26.5 on page 321
- “relative-position” — § 7.18.4 on page 272
- “score-spaces” — § 7.26.6 on page 321
- “text-decoration” — § 7.14.4 on page 253
- “text-shadow” — § 7.14.5 on page 254
- “text-transform” — § 7.14.6 on page 255
- “word-spacing” — § 7.14.8 on page 256

6.7. Formatting Objects for Tables

6.7.1. Introduction

There are nine formatting objects used to construct tables: fo:table-and-caption, fo:table, fo:table-column, fo:table-caption, fo:table-header, fo:table-footer, fo:table-body, fo:table-row, and fo:table-cell. The result tree structure is shown below.



6.7.1.1. Examples

6.7.1.1.1. Simple Table with CALS Column-width Specifications

This example is using a simple, "CALS-like", markup for the table elements. The column-widths are specified using the full CALS column-width specification.

Input sample:

```

<doc>
<table>
<tgroup cols="3">
<colspec colname="col1" colwidth="1*" />
<colspec colname="col2" colwidth="2*+2pi" />
<colspec colname="col3" colwidth="72" />
<tbody>
<row>
<entry colnum="1" valign="top"><p>Cell 1</p></entry>
<entry colnum="2" valign="middle" align="center"><p>Cell 2</p></entry>
<entry colnum="3" align="center"><p>Cell 3</p></entry>
</row>
</tbody>
</tgroup>
</table>
</doc>

```

XSL Stylesheet:

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

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>
<xsl:template match="p">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="table">
  <fo:table width="12cm" table-layout="fixed">
    <xsl:apply-templates/>
  </fo:table>
</xsl:template>
<xsl:template match="colspec">
  <fo:table-column>
    <xsl:attribute name="column-number">
      <xsl:number count="colspec"/>
    </xsl:attribute>
    <xsl:attribute name="column-width">
      <xsl:call-template name="calc.column.width">
        <xsl:with-param name="colwidth">
          <xsl:value-of select="@colwidth"/>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:attribute>
  </fo:table-column>
</xsl:template>
<xsl:template match="tbody">
  <fo:table-body>
    <xsl:apply-templates/>
  </fo:table-body>
</xsl:template>
<xsl:template match="row">
  <fo:table-row>
    <xsl:apply-templates/>
  </fo:table-row>
</xsl:template>
<xsl:template match="entry">
  <fo:table-cell column-number="{@colnum}">
    <xsl:if test="@valign">
      <xsl:choose>
        <xsl:when test="@valign='middle'">
          <xsl:attribute name="display-align">center</xsl:attribute>
        </xsl:when>
        <xsl:otherwise>
          <xsl:attribute name="display-align">
            <xsl:value-of select="@valign"/>
          </xsl:attribute>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:if>
    <xsl:if test="@align">
      <xsl:attribute name="text-align">

```

```

        <xsl:value-of select="@align"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </fo:table-cell>
</xsl:template>
<xsl:template name="calc.column.width">
<!-- **
  * <p>Calculate an XSL FO table column-width specification from a
  * CALS table column-width specification.</p>
  *
  * <p>CALS expresses table column-widths in the following basic
  * forms:</p>
  *
  * <ul>
  * <li><b>99.99units</b>, a fixed length-specifier.</li>
  * <li><b>99.99</b>, a fixed length-specifier without any units.</li>
  * <li><b>99.99*</b>, a relative length-specifier.</li>
  * <li><b>99.99*+99.99units</b>, a combination of both.</li>
  * </ul>
  *
  * <p>The CALS units are points (pt), picas (pi), centimeters (cm),
  * millimeters (mm), and inches (in). These are the same units as XSL,
  * except that XSL abbreviates picas "pc" instead of "pi". If a length
  * specifier has no units, the CALS default unit (pt) is assumed.</p>
  *
  * <p>Relative length-specifiers are represented in XSL with the
  * proportional-column-width() function.</p>
  *
  * <p>Here are some examples:</p>
  *
  * <ul>
  * <li>"36pt" becomes "36pt"</li>
  * <li>"3pi" becomes "3pc"</li>
  * <li>"36" becomes "36pt"</li>
  * <li>"3*" becomes "proportional-column-width(3)"</li>
  * <li>"3*+2pi" becomes "proportional-column-width(3)+2pc"</li>
  * <li>"1*+2" becomes "proportional-column-width(1)+2pt"</li>
  * </ul>
  *
  * @param colwidth The CALS column width specification.
  *
  * @returns The XSL column width specification.
  * -->
<xsl:param name="colwidth">1*</xsl:param>
<!-- Ok, the colwidth could have any one of the following forms: -->
<!--      1*      = proportional width -->
<!--      1unit   = 1.0 units wide -->
<!--      1       = 1pt wide -->
<!--      1*+1unit = proportional width + some fixed width -->
<!--      1*+1    = proportional width + some fixed width -->
<!-- If it has a proportional width, translate it to XSL -->
<xsl:if test="contains($colwidth, '*')">

```

```

    <xsl:text>proportional-column-width(</xsl:text>
    <xsl:value-of select="substring-before($colwidth, '*')"/>
    <xsl:text>)</xsl:text>
</xsl:if>
<!-- Now get the non-proportional part of the specification -->
<xsl:variable name="width-units">
  <xsl:choose>
    <xsl:when test="contains($colwidth, '*')">
      <xsl:value-of
        select="normalize-space(substring-after($colwidth, '*'))"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="normalize-space($colwidth)"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<!-- Now the width-units could have any one of the following forms: -->
<!--           = <empty string> -->
<!--      lunit      = 1.0 units wide -->
<!--      1          = 1pt wide -->
<!-- with an optional leading sign -->
<!-- Get the width part by blanking out the units part and discarding -->
<!-- whitespace. -->
<xsl:variable name="width"
  select="normalize-space(translate($width-units,
                                   '+-0123456789.abcdefghijklmnopqrstuvwxyz',
                                   '+-0123456789.'))"/>
<!-- Get the units part by blanking out the width part and discarding -->
<!-- whitespace. -->
<xsl:variable name="units"
  select="normalize-space(translate($width-units,
                                   'abcdefghijklmnopqrstuvwxyz+-0123456789.',
                                   'abcdefghijklmnopqrstuvwxyz'))"/>
<!-- Output the width -->
<xsl:value-of select="$width"/>
<!-- Output the units, translated appropriately -->
<xsl:choose>
  <xsl:when test="$units = 'pi'">pc</xsl:when>
  <xsl:when test="$units = '' and $width != ''">pt</xsl:when>
  <xsl:otherwise><xsl:value-of select="$units"/></xsl:otherwise>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>

```

Result Instance: elements and attributes in the fo: namespace

```

<fo:table width="12cm" table-layout="fixed">
  <fo:table-column column-number="1" column-width="proportional-column-width(1)">
  </fo:table-column>
  <fo:table-column column-number="2"
column-width="proportional-column-width(2)+2pc">
  </fo:table-column>
  <fo:table-column column-number="3" column-width="72pt">
  </fo:table-column>

```



```

<fo:table-body>
  <fo:table-row>
    <fo:table-cell column-number="1" display-align="top">
      <fo:block>Cell 1
    </fo:block>
    </fo:table-cell>
    <fo:table-cell column-number="2" display-align="center" text-align="center">
      <fo:block>Cell 2
    </fo:block>
    </fo:table-cell>
    <fo:table-cell column-number="3" text-align="center">
      <fo:block>Cell 3
    </fo:block>
    </fo:table-cell>
  </fo:table-row>
</fo:table-body>
</fo:table>

```

6.7.2. fo:table-and-caption

Common Usage:

The fo:table-and-caption flow object is used for formatting a table together with its caption.

NOTE: A fo:table-and-caption may be placed inline by enclosing it in an fo:inline-container.

NOTE: This formatting object corresponds to the CSS anonymous box that encloses the table caption and the table.

Areas:

The fo:table-and-caption formatting object generates one or more *normal block-areas*. The fo:table-and-caption returns these areas, any *page-level-out-of-line* areas, and any *reference-level-out-of-line* areas returned by the children of the fo:table-and-caption.

Constraints:

No area may have more than one normal child area returned by the same fo:table-and-caption formatting object.

The children of the areas generated by the fo:table-and-caption are one or two areas; one for the table caption and one for the table itself. These are positioned relative to each other as specified by the *caption-side* trait. They are placed relative to the content-rectangle of the generated area as specified by the *text-align* trait.

Contents:

([table-caption?](#), [table](#))

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172

- [Common Margin Properties-Block](#) — § 7.9 on page 206
- “caption-side” — § 7.24.7 on page 300
- “id” — § 7.26.2 on page 320
- “keep-together” — § 7.17.3 on page 265
- “keep-with-next” — § 7.17.4 on page 266
- “keep-with-previous” — § 7.17.5 on page 267
- “relative-position” — § 7.18.4 on page 272

6.7.3. fo:table

Common Usage:

The fo:table flow object is used for formatting the tabular material of a table.

The fo:table flow object and its child flow objects model the visual layout of a table in a "row primary" manner. A complete table may be seen as consisting of a grid of rows and columns where each cell occupies one or more grid units in the row-progression-direction and column-progression-direction.

The table content is divided into a header (optional), footer (optional), and one or more bodies. Properties specify if the headers and footers should be repeated at a break in the table. Each of these parts occupies one or more rows in the table grid.

Areas:

The fo:table formatting object generates and returns one or more *normal block-areas*. In addition the fo:table returns any *page-level-out-of-line* areas, and any *reference-level-out-of-line* areas returned by the children of the fo:table.

The areas generated and returned by the fo:table formatting object have as children:

- Areas, with only background, corresponding to the column-groups, columns, and rows.
- Areas returned by the fo:table-cell formatting objects.

These areas have a z-index controlling the rendering order determined in accordance with 17.5.1 of the CSS2 specification.

NOTE: A cell that is spanned may have a different background in each of the grid units it occupies.

Trait Derivation:

The areas generated and returned by the fo:table formatting object have a value of "true" for the *is-reference-area*.

The column-progression-direction and row-progression-direction are determined by the *writing-mode* trait. Columns use the inline-progression-direction, and rows use the block-progression-direction.

The method for deriving the border traits for a table is specified by the "border-collapse" property.

If the value of the "border-collapse" property is "separate" the border is composed of two components. The first, which is placed with the inside edge coincident with the outermost table grid boundary line, has the width of half the value for the "border-separation" property. It is filled in accordance with the "background" property of the fo:table. Outside the outermost table grid boundary line is placed, for each side of the table, a border based on a border specified on the table.

If the value of the "border-collapse" property is "collapse" the border is determined, for each segment, at the cell level.

NOTE: By specifying an appropriately high precedence on the border specification for the `fo:table` one may ensure that this specification is the one used on all border segments.

Constraints:

No area may have more than one normal child area returned by the same `fo:table` formatting object.

The `inline-progression-dimension` of the `content-rectangle` of the table is the sum of the `inline-progression-dimensions` of the columns in the table grid. The method used to determine these `inline-progression-dimensions` is governed by the values of the `table-layout` and the `inline-progression-dimension` traits in the following manner:

inline-progression-dimension="auto" table-layout="auto"

The automatic table layout shall be used.

inline-progression-dimension="auto" table-layout="fixed"

The automatic table layout shall be used.

inline-progression-dimension=<length> or <percentage> table-layout="auto"

The automatic table layout shall be used.

inline-progression-dimension=<length> or <percentage> table-layout="fixed"

The fixed table layout shall be used.

The automatic table layout and fixed table layout is defined in 17.5.2 of the CSS2 specification.

The method for determining the `block-progression-dimension` of the table is governed by the `block-progression-dimension` trait.

NOTE: The CSS2 specification explicitly does not specify what the behavior should be if there is a mismatch between an explicitly specified table `block-progression-dimension` and the `block-progression-dimensions` of the content.

NOTE: The use of the `"proportional-column-width()"` function is only permitted when the fixed table layout is used.

If the use of proportional column widths are desired on a table of an unknown explicit width, the `inline-progression-dimension` cannot be specified to be `"auto"`. Instead, the width must be specified as a percentage. For example, setting `table-layout="fixed"` and `inline-progression-dimension="100%"` would allow proportional column widths while simultaneously creating a table as wide as possible in the current context.

NOTE: The result of using a percentage for the width may be unpredictable, especially when using the automatic table layout.

Contents:

(`table-column*`, `table-header?`, `table-footer?`, `table-body+`)

In addition this formatting object may have a sequence of zero or more `fo:markers` as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166

- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Block](#) — § 7.9 on page 206
- [“block-progression-dimension”](#) — § 7.12.1 on page 230
- [“border-after-precedence”](#) — § 7.24.1 on page 298
- [“border-before-precedence”](#) — § 7.24.2 on page 298
- [“border-collapse”](#) — § 7.24.3 on page 299
- [“border-end-precedence”](#) — § 7.24.4 on page 299
- [“border-separation”](#) — § 7.24.5 on page 300
- [“border-start-precedence”](#) — § 7.24.6 on page 300
- [“break-after”](#) — § 7.17.1 on page 264
- [“break-before”](#) — § 7.17.2 on page 265
- [“id”](#) — § 7.26.2 on page 320
- [“inline-progression-dimension”](#) — § 7.12.5 on page 234
- [“height”](#) — § 7.12.4 on page 233
- [“keep-together”](#) — § 7.17.3 on page 265
- [“keep-with-next”](#) — § 7.17.4 on page 266
- [“keep-with-previous”](#) — § 7.17.5 on page 267
- [“relative-position”](#) — § 7.18.4 on page 272
- [“table-layout”](#) — § 7.24.16 on page 305
- [“table-omit-footer-at-break”](#) — § 7.24.17 on page 306
- [“table-omit-header-at-break”](#) — § 7.24.18 on page 306
- [“width”](#) — § 7.12.12 on page 239
- [“writing-mode”](#) — § 7.25.7 on page 317

6.7.4. fo:table-column

Common Usage:

The fo:table-column auxiliary formatting object specifies characteristics applicable to table cells that have the same column and span. The most important property is the "column-width" property.

Areas:

The fo:table-column formatting object does not generate or return any areas. It holds a set of traits that provide constraints on the column widths and a specification of some presentation characteristics, such as background which affects the areas generated by the fo:table (see § 6.7.3 – fo:table on page 122). Inheritable properties may also be specified on the fo:table-column. These can be referenced by the from-table-column() function in an expression.

Constraints:

None.

Contents:

EMPTY

The following properties apply to this formatting object:

- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
Only the background properties from this set applies.
- [“column-number”](#) — § 7.24.8 on page 302
- [“column-width”](#) — § 7.24.9 on page 302
- [“number-columns-repeated”](#) — § 7.24.12 on page 304

- “[number-columns-spanned](#)” — § 7.24.13 on page 304
- “[visibility](#)” — § 7.26.8 on page 322

6.7.5. [fo:table-caption](#)

Common Usage:

The `fo:table-caption` formatting object is used to contain block-level formatting objects containing the caption for the table only when using the `fo:table-and-caption`.

Areas:

The `fo:table-caption` formatting object generates one or more *normal reference-areas*. The `fo:table-caption` returns these reference-areas and any *page-level-out-of-line* areas returned by the children of the `fo:table-caption`.

Trait Derivation:

The areas generated by the `fo:table-caption` formatting object have a value of “true” for the *is-reference-area*.

Constraints:

For the case when the value of the *caption-side* trait is “before” or “after” the inline-progression-dimension of the content-rectangle of the generated reference-area is equal to the inline-progression-dimension of the content-rectangle of the reference-area that encloses it.

When the value is “start” or “end” the inline-progression-dimension of the generated reference-area is constrained by the value of the *width* trait.

When the value is “top”, “bottom”, “left”, or “right” the value is mapped in the same way as for corresponding properties (see § 5.3 – [Computing the Values of Corresponding Properties](#) on page 42) and the property is then treated as if the corresponding value had been specified.

If the caption is to be positioned before the table, the areas generated by the `fo:table-caption` shall be placed in the area tree as though the `fo:table-caption` had a “keep-with-next” property with value “0”.

If the caption is to be positioned after the table, the areas generated by the `fo:table-caption` shall be placed in the area tree as though the `fo:table-caption` had a “keep-with-next” property with value “0”.

No area may have more than one normal child area returned by the same `fo:table-caption` formatting object.

The children of each normal area returned by an `fo:table-caption` formatting object must be normal *block-areas* returned by the children of the `fo:table-caption`, must be *properly stacked*, and must be *properly ordered*.

Any *reference-level-out-of-line* areas returned by the children of the `fo:table-caption` are handled as described in § 6.10.2 – [fo:float](#) on page 152.

Contents:

([%block;](#)) +

In addition this formatting object may have a sequence of zero or more `fo:markers` as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166

- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- “[block-progression-dimension](#)” — § 7.12.1 on page 230
- “[height](#)” — § 7.12.4 on page 233
- “[id](#)” — § 7.26.2 on page 320
- “[inline-progression-dimension](#)” — § 7.12.5 on page 234
- “[keep-together](#)” — § 7.17.3 on page 265
- “[relative-position](#)” — § 7.18.4 on page 272
- “[width](#)” — § 7.12.12 on page 239

6.7.6. fo:table-header

Common Usage:

The fo:table-header formatting object is used to contain the content of the table header.

Areas:

The fo:table-header formatting object does not generate any areas. The fo:table-header formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:table-header.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:table-header is the same order as the children are ordered under the fo:table-header.

Contents:

([table-row](#)+ | [table-cell](#)+)

The fo:table-header has fo:table-row (one or more) as its children, or alternatively fo:table-cell (one or more). In the latter case cells are grouped into rows using the starts-row and ends-row properties.

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- “[id](#)” — § 7.26.2 on page 320
- “[relative-position](#)” — § 7.18.4 on page 272

6.7.7. fo:table-footer

Common Usage:

The fo:table-footer formatting object is used to contain the content of the table footer.

Areas:

The fo:table-footer formatting object does not generate any areas. The fo:table-footer formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:table-footer.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:table-footer is the same order as the children are ordered under the fo:table-footer.

Contents:

([table-row+](#) | [table-cell+](#))

The fo:table-footer has fo:table-row (one or more) as its children, or alternatively fo:table-cell (one or more). In the latter case cells are grouped into rows using the starts-row and ends-row properties.

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- “id” — § 7.26.2 on page 320
- “relative-position” — § 7.18.4 on page 272

6.7.8. fo:table-body

Common Usage:

The fo:table-body formatting object is used to contain the content of the table body.

Areas:

The fo:table-body formatting object does not generate any areas. The fo:table-body formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:table-body.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:table-body is the same order as the children are ordered under the fo:table-body.

Contents:

([table-row+](#) | [table-cell+](#))

The fo:table-body has fo:table-row (one or more) as its children, or alternatively fo:table-cell (one or more). In the latter case cells are grouped into rows using the starts-row and ends-row properties.

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- “id” — § 7.26.2 on page 320
- “relative-position” — § 7.18.4 on page 272

6.7.9. fo:table-row

Common Usage:

The fo:table-row formatting object is used to group table-cells into rows; all table-cells in a table-row start in the same geometric row on the table grid.

Areas:

The `fo:table-row` formatting object does not generate any areas. The `fo:table-row` formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the `fo:table-row`. The `fo:table-row` holds a specification of some presentation characteristics, such as background which affects the areas generated by the `fo:table` (see § 6.7.3 – `fo:table` on page 122).

Constraints:

The order of concatenation of the sequences of areas returned by the children of the `fo:table-row` is the same order as the children are ordered under the `fo:table-row`.

The method for determining the height of the row in the grid is governed by the *row-height* trait.

Contents:

(`table-cell`+)

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
 - [“block-progression-dimension”](#) — § 7.12.1 on page 230
 - [Common Aural Properties](#) — § 7.5 on page 166
 - [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- Only the background properties from this set applies.
- [“break-after”](#) — § 7.17.1 on page 264
 - [“break-before”](#) — § 7.17.2 on page 265
 - [“id”](#) — § 7.26.2 on page 320
 - [“height”](#) — § 7.12.4 on page 233
 - [“keep-together”](#) — § 7.17.3 on page 265
 - [“keep-with-next”](#) — § 7.17.4 on page 266
 - [“keep-with-previous”](#) — § 7.17.5 on page 267
 - [“relative-position”](#) — § 7.18.4 on page 272

6.7.10. `fo:table-cell`*Common Usage:*

The `fo:table-cell` formatting object is used to group content to be placed in a table cell.

The `"starts-row"` and `"ends-row"` properties can be used when the input data does not have elements containing the cells in each row, but instead, for example, each row starts at elements of a particular type.

Areas:

The `fo:table-cell` formatting object generates one or more *normal reference-areas*. The `fo:table-cell` returns these reference-areas and any *page-level-out-of-line* areas returned by the children of the `fo:table-cell`.

Trait Derivation:

The areas generated by the `fo:table-cell` formatting object have a value of `"true"` for the *is-reference-area*.

The method for deriving the border for a cell is specified by the *border-collapse* trait.

If the value of the *border-collapse* trait is `"separate"` the border is composed of two components. The

first, which is placed with the outside edge coincident with the table grid boundary line, has the width of half the value for the *border-separation* trait. It is filled in accordance with the *background* trait of the *fo:table*. Inside this border is placed, for each side of the cell, a border based on a border specified on the cell or inherited.

If the value of the *border-collapse* trait is "collapse" the border for each side of the cell is determined by, for each segment of a border, selecting, from all border specifications for that segment, the border that has the highest precedence. It is an error if there are two such borders that have the same precedence but are not identical. Each border segment is placed centered on the table grid boundary line.

Constraints:

A table-cell occupies one or more grid units in the row-progression-direction and column-progression-direction. The content-rectangle of the cell is the size of the portion of the grid the cell occupies minus, for each of the four sides:

- If the value of the *border-collapse* trait is "separate": half the value of the *border-separation* trait; otherwise 0.
- If the value of the *border-collapse* trait is "separate": the thickness of the cell-border; otherwise half the thickness of the effective border.
- The cell padding.

The method for determining the block-progression-dimension of the cell in the grid is governed by the *row-height* trait.

No area may have more than one normal child area returned by the same *fo:table-cell* formatting object.

The children of each normal area returned by an *fo:table-cell* formatting object must be normal *block-areas* returned by the children of the *fo:table-cell*, must be *properly stacked*, and must be *properly ordered*.

Any *reference-level-out-of-line* areas returned by the children of the *fo:table-cell* are handled as described in § 6.10.2 – *fo:float* on page 152.

Contents:

(%block;)+

In addition this formatting object may have a sequence of zero or more *fo:markers* as its initial children.

The following properties apply to this formatting object:

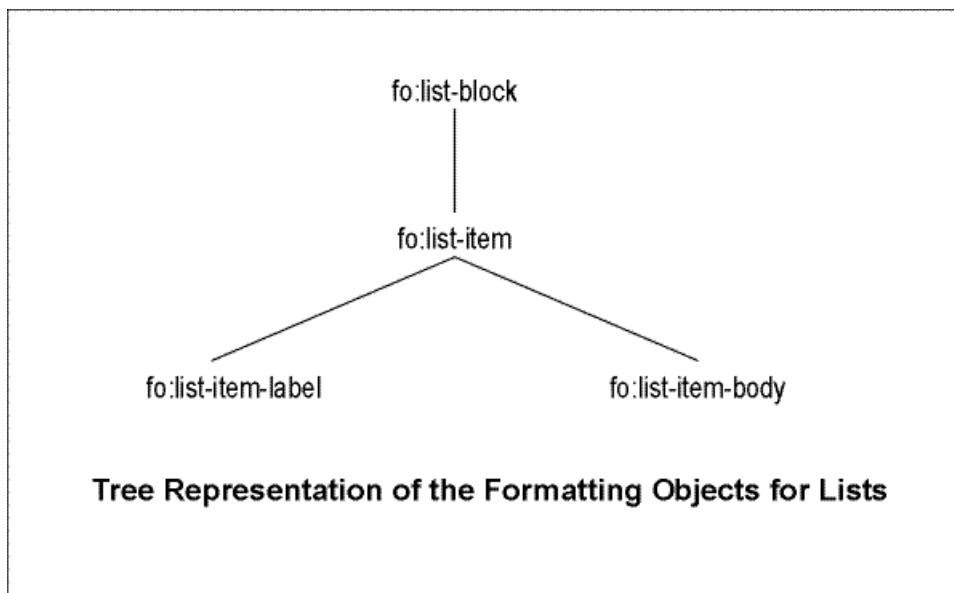
- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- “border-after-precedence” — § 7.24.1 on page 298
- “border-before-precedence” — § 7.24.2 on page 298
- “border-end-precedence” — § 7.24.4 on page 299
- “border-start-precedence” — § 7.24.6 on page 300
- “block-progression-dimension” — § 7.12.1 on page 230
- “column-number” — § 7.24.8 on page 302
- “display-align” — § 7.11.4 on page 226
- “relative-align” — § 7.11.6 on page 229
- “empty-cells” — § 7.24.10 on page 303
- “ends-row” — § 7.24.11 on page 303

- “height” — § 7.12.4 on page 233
- “id” — § 7.26.2 on page 320
- “number-columns-spanned” — § 7.24.13 on page 304
- “number-rows-spanned” — § 7.24.14 on page 305
- “relative-position” — § 7.18.4 on page 272
- “starts-row” — § 7.24.15 on page 305
- “width” — § 7.12.12 on page 239

6.8. Formatting Objects for Lists

6.8.1. Introduction

There are four formatting objects used to construct lists: `fo:list-block`, `fo:list-item`, `fo:list-item-label`, and `fo:list-item-body`.



Tree representation of the formatting Objects for Lists.

The `fo:list-block` has the role of containing the complete list and of specifying values used for the list geometry in the inline-progression-direction (see details below).

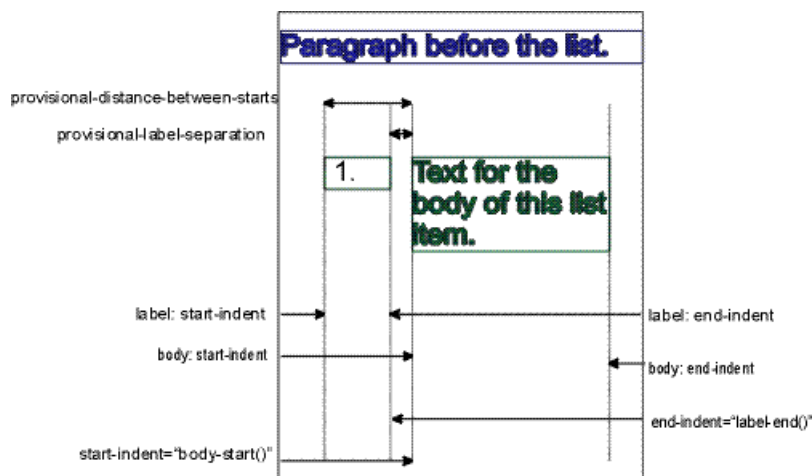
The children of the `fo:list-block` are one or more `fo:list-item`, each containing a pair of `fo:list-item-label` and `fo:list-item-body`.

The `fo:list-item` has the role of containing each item in a list.

The `fo:list-item-label` has the role of containing the content, block-level formatting objects, of the label for the list-item; typically an `fo:block` containing a number, a dingbat character, or a term.

The `fo:list-item-body` has the role of containing the content, block-level formatting objects, of the body of the list-item; typically one or more `fo:block`.

The placement, in the block-progression-direction, of the label with respect to the body is made in accordance with the "vertical-align" property of the `fo:list-item`.



The specification of the list geometry in the inline-progression-direction is achieved by:

- Specifying appropriate values of the "provisional-distance-between-starts" and "provisional-label-separation" properties. The "provisional-distance-between-starts" specifies the desired distance between the start-indents of the label and the body of the list-item. The "provisional-label-separation" specifies the desired separation between the end-indent of the label and the start-indent of the body of the list-item.
- Specifying `end-indent="label-end()"` on the `fo:list-item-label`.
Specifying `start-indent="body-start()"` on the `fo:list-item-body`.

NOTE: These list specific functions are defined below.

The start-indent of the list-item-label and end-indent of the list-item-body, if desired, are typically specified as a length.

6.8.1.1. Examples

6.8.1.1.1. Enumerated List

The list-items are contained in an "ol" element. The items are contained in "item" elements and contain text (as opposed to paragraphs).

The style is to enumerate the items alphabetically with a dot after the letter.

Input sample:

```
<ol>
<item>List item 1.</item>
<item>List item 2.</item>
<item>List item 3.</item>
</ol>
```

XSL Stylesheet:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version='1.0'>
<xsl:template match="ol">
```

```

<fo:list-block provisional-distance-between-starts="15mm"
  provisional-label-separation="5mm">
  <xsl:apply-templates/>
</fo:list-block>
</xsl:template>
<xsl:template match="ol/item">
  <fo:list-item>
    <fo:list-item-label start-indent="5mm" end-indent="label-end(">
      <fo:block>
        <xsl:number format="a."/>
      </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start(">
      <fo:block>
        <xsl:apply-templates/>
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
</xsl:stylesheet>

```

Result Instance: elements and attributes in the fo: namespace

```

<fo:list-block provisional-distance-between-starts="15mm"
  provisional-label-separation="5mm">
  <fo:list-item>
    <fo:list-item-label start-indent="5mm" end-indent="label-end(">
      <fo:block>a.
    </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start(">
      <fo:block>List item 1.
    </fo:block>
    </fo:list-item-body>
  </fo:list-item>
  <fo:list-item>
    <fo:list-item-label start-indent="5mm" end-indent="label-end(">
      <fo:block>b.
    </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start(">
      <fo:block>List item 2.
    </fo:block>
    </fo:list-item-body>
  </fo:list-item>
  <fo:list-item>
    <fo:list-item-label start-indent="5mm" end-indent="label-end(">
      <fo:block>c.
    </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start(">
      <fo:block>List item 3.
    </fo:block>
    </fo:list-item-body>
  </fo:list-item>

```

```
</fo:list-item>
</fo:list-block>
```

6.8.1.1.2. HTML-style "dl" lists

In this example the stylesheet processes HTML-style "dl" lists, which contain unwrapped pairs of "dt" and "dd" elements, transforming them into fo:list-blocks.

Balanced pairs of "dt"/"dd"s are converted into fo:list-items. For unbalanced "dt"/"dd"s, the stylesheet makes the following assumptions:

- Multiple "dt"s are grouped together into a single fo:list-item-label in a single list-item.
- Multiple DDs are:
 - Output as individual FO list-items with an empty list-item-label if the stylesheet variable \$allow-naked-dd is true.
 - Are grouped together into a single FO list-item-body if \$allow-naked-dd is false.

In other words, given a structure like this:

```
<doc>
<dl>
  <dt>term</dt>
  <dd>definition</dd>
  <dt>term</dt>
  <dt>term</dt>
  <dd>definition</dd>
  <dt>term</dt>
  <dd>definition</dd>
  <dd>definition</dd>
</dl>
</doc>
```

If \$allow-naked-dd is true, the result instance: elements and attributes in the fo: namespace is:

```
<fo:list-block provisional-distance-between-starts="35mm"
  provisional-label-separation="5mm">
  <fo:list-item>
    <fo:list-item-label end-indent="label-end()">
      <fo:block>term
    </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block>definition
    </fo:block>
    </fo:list-item-body>
  </fo:list-item>
  <fo:list-item>
    <fo:list-item-label end-indent="label-end()">
      <fo:block>term
    </fo:block>
    <fo:block>term
    </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
```

```

    <fo:block>definition
  </fo:block>
</fo:list-item-body>
</fo:list-item>
<fo:list-item>
  <fo:list-item-label end-indent="label-end()">
    <fo:block>term
  </fo:block>
</fo:list-item-label>
  <fo:list-item-body start-indent="body-start()">
    <fo:block>definition
  </fo:block>
</fo:list-item-body>
</fo:list-item>
<fo:list-item>
  <fo:list-item-label end-indent="label-end()">
  </fo:list-item-label>
  <fo:list-item-body start-indent="body-start()">
    <fo:block>definition
  </fo:block>
</fo:list-item-body>
</fo:list-item>
</fo:list-block>

```

If \$allow-naked-dd is false, the result instance: elements and attributes in the fo: namespace is:

```

<fo:list-block provisional-distance-between-starts="35mm"
  provisional-label-separation="5mm">
  <fo:list-item>
    <fo:list-item-label end-indent="label-end()">
      <fo:block>term
    </fo:block>
  </fo:list-item-label>
  <fo:list-item-body start-indent="body-start()">
    <fo:block>definition
  </fo:block>
</fo:list-item-body>
</fo:list-item>
<fo:list-item>
  <fo:list-item-label end-indent="label-end()">
    <fo:block>term
  </fo:block>
  <fo:block>term
  </fo:block>
</fo:list-item-label>
  <fo:list-item-body start-indent="body-start()">
    <fo:block>definition
  </fo:block>
</fo:list-item-body>
</fo:list-item>
<fo:list-item>
  <fo:list-item-label end-indent="label-end()">
    <fo:block>term
  </fo:block>
</fo:block>

```

```

    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start(">
      <fo:block>definition
    </fo:block>
    <fo:block>definition
    </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</fo:list-block>

```

XSL Stylesheet:

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>
<xsl:include href="dtdd.xsl"/>
<xsl:template match="doc">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="dl">
  <xsl:call-template name="process.dl"/>
</xsl:template>
<xsl:template match="dt|dd">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
</xsl:stylesheet>

```

Included stylesheet "dtdd.xsl"

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>
<xsl:variable name="allow-naked-dd" select="true()"/>
<xsl:template name="process.dl">
  <fo:list-block provisional-distance-between-starts="35mm"
    provisional-label-separation="5mm">
    <xsl:choose>
      <xsl:when test="$allow-naked-dd">
        <xsl:call-template name="process.dl.content.with.naked.dd"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:call-template name="process.dl.content"/>
      </xsl:otherwise>
    </xsl:choose>
  </fo:list-block>
</xsl:template>
<xsl:template name="process.dl.content.with.naked.dd">
  <xsl:param name="dts" select="./force-list-to-be-empty"/>
  <xsl:param name="nodes" select="*" />
  <xsl:choose>
    <xsl:when test="count($nodes)=0">

```

```

    <!-- Out of nodes, output any pending DTs -->
    <xsl:if test="count($dts)>0">
      <fo:list-item>
        <fo:list-item-label end-indent="label-end()">
          <xsl:apply-templates select="$dts"/>
        </fo:list-item-label>
        <fo:list-item-body start-indent="body-start()"/>
      </fo:list-item>
    </xsl:if>
  </xsl:when>
  <xsl:when test="name($nodes[1])='dd'">
    <!-- We found a DD, output the DTs and the DD -->
    <fo:list-item>
      <fo:list-item-label end-indent="label-end()">
        <xsl:apply-templates select="$dts"/>
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">
        <xsl:apply-templates select="$nodes[1]"/>
      </fo:list-item-body>
    </fo:list-item>
    <xsl:call-template name="process.dl.content.with.naked.dd">
      <xsl:with-param name="nodes" select="$nodes[position()>1]"/>
    </xsl:call-template>
  </xsl:when>
  <xsl:when test="name($nodes[1])='dt'">
    <!-- We found a DT, add it to the list of DTs and loop -->
    <xsl:call-template name="process.dl.content.with.naked.dd">
      <xsl:with-param name="dts" select="$dts|$nodes[1]"/>
      <xsl:with-param name="nodes" select="$nodes[position()>1]"/>
    </xsl:call-template>
  </xsl:when>
  <xsl:otherwise>
    <!-- This shouldn't happen -->
    <xsl:message>
      <xsl:text>DT/DD list contained something bogus (</xsl:text>
      <xsl:value-of select="name($nodes[1])"/>
      <xsl:text>).</xsl:text>
    </xsl:message>
  </xsl:otherwise>
</xsl:choose>
</xsl:template>
<xsl:template name="process.dl.content">
  <xsl:param name="dts" select="./force-list-to-be-empty"/>
  <xsl:param name="dds" select="./force-list-to-be-empty"/>
  <xsl:param name="output-on"></xsl:param>
  <xsl:param name="nodes" select="*" />
  <!-- The algorithm here is to build up a list of DTs and DDs, -->
  <!-- outputting them only on the transition from DD back to DT -->
  <xsl:choose>
    <xsl:when test="count($nodes)=0">
      <!-- Out of nodes, output any pending elements -->
      <xsl:if test="count($dts)>0 or count($dds)>0">
        <fo:list-item>

```

```

        <fo:list-item-label end-indent="label-end()">
            <xsl:apply-templates select="$dts"/>
        </fo:list-item-label>
        <fo:list-item-body start-indent="body-start()">
            <xsl:apply-templates select="$dds"/>
        </fo:list-item-body>
    </fo:list-item>
</xsl:if>
</xsl:when>
<xsl:when test="name($nodes[1])=$output-on">
    <!-- We're making the transition from DD back to DT -->
    <fo:list-item>
        <fo:list-item-label end-indent="label-end()">
            <xsl:apply-templates select="$dts"/>
        </fo:list-item-label>
        <fo:list-item-body start-indent="body-start()">
            <xsl:apply-templates select="$dds"/>
        </fo:list-item-body>
    </fo:list-item>
    <!-- Reprocess this node (and the rest of the node list) -->
    <!-- resetting the output-on state to nil -->
    <xsl:call-template name="process.dl.content">
        <xsl:with-param name="nodes" select="$nodes"/>
    </xsl:call-template>
</xsl:when>
<xsl:when test="name($nodes[1])='dt'">
    <!-- We found a DT, add it to the list and loop -->
    <xsl:call-template name="process.dl.content">
        <xsl:with-param name="dts" select="$dts|$nodes[1]"/>
        <xsl:with-param name="dds" select="$dds"/>
        <xsl:with-param name="nodes" select="$nodes[position()>1]"/>
    </xsl:call-template>
</xsl:when>
<xsl:when test="name($nodes[1])='dd'">
    <!-- We found a DD, add it to the list and loop, noting that -->
    <!-- the next time we cross back to DT's, we need to output the -->
    <!-- current DT/DDs. -->
    <xsl:call-template name="process.dl.content">
        <xsl:with-param name="dts" select="$dts"/>
        <xsl:with-param name="dds" select="$dds|$nodes[1]"/>
        <xsl:with-param name="output-on">dt</xsl:with-param>
        <xsl:with-param name="nodes" select="$nodes[position()>1]"/>
    </xsl:call-template>
</xsl:when>
<xsl:otherwise>
    <!-- This shouldn't happen -->
    <xsl:message>
        <xsl:text>DT/DD list contained something bogus (</xsl:text>
        <xsl:value-of select="name($nodes[1])"/>
        <xsl:text>).</xsl:text>
    </xsl:message>
</xsl:otherwise>
</xsl:choose>

```

```
</xsl:template>
</xsl:stylesheet>
```

The "dtdd.xsl" stylesheet may be customized in the following ways:

- Set the value of \$allow-naked-dd to control the processing of unbalanced "dd"s.
- Change "dt" to the name of the element which is a term in the list.
- Change "dd" to the name of the element which is a definition in the list.
- In the, perhaps unlikely, event that the documents may contain an element named "force-list-to-be-empty", that element name should be changed to a name that is not used in the documents.

In the stylesheet using the "dtdd.xsl" stylesheet change the "dl" to the name of the element which is the wrapper for the list.

6.8.2. fo:list-block

Common Usage:

The fo:list-block flow object is used to format a list.

Areas:

The fo:list-block formatting object generates one or more *normal block-areas*. The fo:list-block returns these areas, any *page-level-out-of-line* areas, and any *reference-level-out-of-line* areas returned by the children of the fo:list-block.

Constraints:

No area may have more than one normal child area returned by the same fo:list-block formatting object.

The children of each normal area returned by an fo:list-block formatting object must be normal *block-areas* returned by the children of the fo:list-block, must be *properly stacked*, and must be *properly ordered*.

Contents:

([list-item](#)*)

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Block](#) — § 7.9 on page 206
- "break-after" — § 7.17.1 on page 264
- "break-before" — § 7.17.2 on page 265
- "id" — § 7.26.2 on page 320
- "keep-together" — § 7.17.3 on page 265
- "keep-with-next" — § 7.17.4 on page 266
- "keep-with-previous" — § 7.17.5 on page 267
- "provisional-distance-between-starts" — § 7.26.4 on page 321
- "provisional-label-separation" — § 7.26.3 on page 320
- "relative-position" — § 7.18.4 on page 272

6.8.3. fo:list-item

Common Usage:

The fo:list-item formatting object contains the label and the body of an item in a list.

Areas:

The fo:list-item formatting object generates one or more *normal block-areas*. The fo:list-item returns these areas, any *page-level-out-of-line* areas, and any *reference-level-out-of-line* areas returned by the children of the fo:list-item.

Constraints:

No area may have more than one normal child area returned by the same fo:list-item formatting object.

The children of each normal area returned by an fo:list-block formatting object must be normal *block-areas* returned by the fo:list-item-label and the fo:list-item-body flow objects and must be *properly ordered*. Those returned by the fo:list-item-label must be *properly stacked* and those returned by the fo:list-item-body must be *properly stacked*.

The children of each normal area returned by an fo:list-block formatting object returned by the fo:list-item-label and fo:list-item-body objects are positioned in the block-progression-direction with respect to each other according to the *relative-align* trait.

In the inline-progression-direction these areas are positioned in the usual manner for properly stacked areas. It is an error if the content-rectangles of the areas overlap.

NOTE: These areas are not reference-areas, hence the indents on all objects within them are measured relative to the reference-area that holds the content of the fo:list-block.

Contents:

([list-item-label](#), [list-item-body](#))

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Block](#) — § 7.9 on page 206
- “break-after” — § 7.17.1 on page 264
- “break-before” — § 7.17.2 on page 265
- “id” — § 7.26.2 on page 320
- “keep-together” — § 7.17.3 on page 265
- “keep-with-next” — § 7.17.4 on page 266
- “keep-with-previous” — § 7.17.5 on page 267
- “relative-align” — § 7.11.6 on page 229
- “relative-position” — § 7.18.4 on page 272

6.8.4. fo:list-item-body

Common Usage:

The fo:list-item-body formatting object contains the content of the body of a list-item.

Areas:

The fo:list-item-body formatting object does not generate any areas. The fo:list-item-body formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:list-item-body.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:list-item-body is the same order as the children are ordered under the fo:list-item-body.

Contents:

(%block;)+

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- “id” — § 7.26.2 on page 320
- “keep-together” — § 7.17.3 on page 265

6.8.5. fo:list-item-label

Common Usage:

The fo:list-item-label formatting object contains the content of the label of a list-item, typically used to either enumerate, identify, or adorn the list-item's body.

Areas:

The fo:list-item-label formatting object does not generate any areas. The fo:list-item-label formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:list-item-label.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:list-item-label is the same order as the children are ordered under the fo:list-item-label.

Contents:

(%block;)+

In addition this formatting object may have a sequence of zero or more fo:markers as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- “id” — § 7.26.2 on page 320
- “keep-together” — § 7.17.3 on page 265

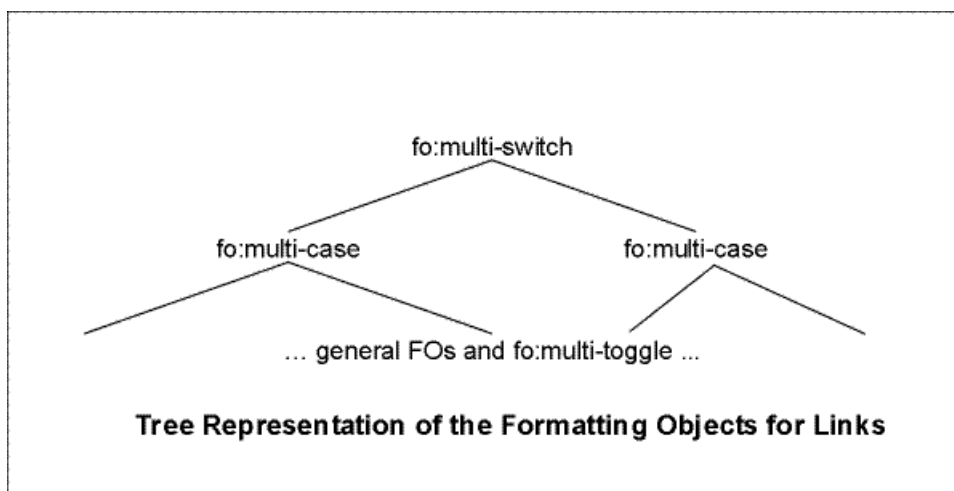
6.9. Dynamic Effects: Link and Multi Formatting Objects

6.9.1. Introduction

Dynamic effects, whereby user actions (including User Agent state) can influence the behavior and/or representation of portions of a document, can be achieved through the use of the formatting objects included in this section:

- One-directional single-target links.
- The ability to switch between the display of two or more formatting object subtrees. This can be used for, e.g., expandable/collapsible table of contents, display of an icon or a full table or graphic.
- The ability to switch between different property values, such as color or font-weight, depending on a User Agent state, such as "hover".

The switching between subtrees is achieved by using the following three formatting objects: fo:multi-switch, fo:multi-case, and fo:multi-toggle. The result tree structure is shown below.



The role of the fo:multi-switch is to wrap fo:multi-case formatting objects, each containing a subtree. Each subtree is given a name on the fo:multi-case formatting object. Activating, for example implemented as clicking on, an fo:multi-toggle causes a named subtree, the previous, the next, or "any" subtree to be displayed; controlled by the "switch-to" property. For "any", an implementation would typically present a list of choices each labeled using the "case-title" property of the fo:multi-case. The initial subtree displayed is controlled by the "starting-state" property on the fo:multi-case.

Switching between different property values is achieved by using the fo:multi-properties and fo:multi-property-set formatting objects, and the merge-property-values() function. For example, an fo:multi-property-set can be used to specify various properties for each of the possible values of the active-state property, and merge-property-values() can be used to apply them on a given formatting object.

6.9.1.1. Examples

6.9.1.1.1. Expandable/Collapsible Table of Contents

Input sample:

```

<doc>
  <chapter><title>Chapter</title>
    <p>Text</p>
    <section><title>Section</title>
      <p>Text</p>
    </section>
    <section><title>Section</title>
      <p>Text</p>
    </section>
  </chapter>
</doc>

```

```

</chapter>
<chapter><title>Chapter</title>
  <p>Text</p>
  <section><title>Section</title>
  <p>Text</p>
</section>
<section><title>Section</title>
  <p>Text</p>
</section>
</chapter>
</doc>

```

In this example the chapter and section titles are extracted into a table of contents placed at the front of the result. The chapter titles are preceded by an icon indicating either collapsed or expanded state. The section titles are only shown in the expanded state. Furthermore, there are links from the titles in the table of contents to the corresponding titles in the body of the document.

The two states are achieved by, for each chapter title, using an `fo:multi-switch` with a `fo:multi-case` for each state. The icon is contained in an `fo:multi-toggle` with the appropriate `fo:multi-case` "switch-to" property to select the other state.

The links in the table of contents are achieved by adding a unique id on the title text in the body of the document and wrapping the title text in the table of contents in an `fo:basic-link` referring to that id.

XSL Stylesheet:

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version='1.0'>
<xsl:template match="doc">
  <!-- create the table of contents -->
  <xsl:apply-templates select="chapter/title" mode="toc"/>
  <!-- do the document -->
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="chapter/title" mode="toc">
  <fo:multi-switch>
    <fo:multi-case case-name="collapsed" case-title="collapsed"
      starting-state="show">
      <fo:block>
        <fo:multi-toggle switch-to="expanded">
          <fo:external-graphic href="plus-icon.gif"/>
        </fo:multi-toggle>
        <fo:basic-link internal-destination="{generate-id(.)}">
          <xsl:number level="multiple" count="chapter" format="1. " />
          <xsl:apply-templates mode="toc"/>
        </fo:basic-link>
      </fo:block>
    </fo:multi-case>
    <fo:multi-case case-name="expanded" case-title="expanded"
      starting-state="hide">
      <fo:block>
        <fo:multi-toggle switch-to="collapsed">
          <fo:external-graphic href="minus-icon.gif"/>

```

```

        </fo:multi-toggle>
        <fo:basic-link internal-destination="{generate-id(.)}">
            <xsl:number level="multiple" count="chapter" format="1. " />
            <xsl:apply-templates mode="toc" />
        </fo:basic-link>
    </fo:block>
    <xsl:apply-templates select="../section/title" mode="toc" />
</fo:multi-case>
</fo:multi-switch>
</xsl:template>
<xsl:template match="section/title" mode="toc">
    <fo:block start-indent="10mm">
        <fo:basic-link internal-destination="{generate-id(.)}">
            <xsl:number level="multiple" count="chapter|section" format="1.1 " />
            <xsl:apply-templates />
        </fo:basic-link>
    </fo:block>
</xsl:template>
<xsl:template match="chapter/title">
    <fo:block id="{generate-id(.)}">
        <xsl:number level="multiple" count="chapter" format="1. " />
        <xsl:apply-templates />
    </fo:block>
</xsl:template>
<xsl:template match="section/title">
    <fo:block id="{generate-id(.)}">
        <xsl:number level="multiple" count="chapter|section" format="1.1 " />
        <xsl:apply-templates />
    </fo:block>
</xsl:template>
<xsl:template match="p">
    <fo:block>
        <xsl:apply-templates />
    </fo:block>
</xsl:template>
</xsl:stylesheet>

```

Result Instance: elements and attributes in the fo: namespace

```

<fo:multi-switch>
  <fo:multi-case case-name="collapsed" case-title="collapsed"
starting-state="show">
    <fo:block>
      <fo:multi-toggle switch-to="expanded">
        <fo:external-graphic href="plus-icon.gif">
          </fo:external-graphic>
        </fo:multi-toggle>
        <fo:basic-link internal-destination="N4">1. Chapter
        </fo:basic-link>
      </fo:block>
    </fo:multi-case>
  <fo:multi-case case-name="expanded" case-title="expanded" starting-state="hide">
    <fo:block>
      <fo:multi-toggle switch-to="collapsed">

```

```

        <fo:external-graphic href="minus-icon.gif">
        </fo:external-graphic>
    </fo:multi-toggle>
    <fo:basic-link internal-destination="N4">1. Chapter
    </fo:basic-link>
</fo:block>
<fo:block start-indent="10mm">
    <fo:basic-link internal-destination="N11">1.1 Section
    </fo:basic-link>
</fo:block>
<fo:block start-indent="10mm">
    <fo:basic-link internal-destination="N19">1.2 Section
    </fo:basic-link>
</fo:block>
</fo:multi-case>
</fo:multi-switch>
<fo:multi-switch>
    <fo:multi-case case-name="collapsed" case-title="collapsed"
starting-state="show">
        <fo:block>
            <fo:multi-toggle switch-to="expanded">
                <fo:external-graphic href="plus-icon.gif">
                </fo:external-graphic>
            </fo:multi-toggle>
            <fo:basic-link internal-destination="N28">2. Chapter
            </fo:basic-link>
        </fo:block>
    </fo:multi-case>
    <fo:multi-case case-name="expanded" case-title="expanded" starting-state="hide">
        <fo:block>
            <fo:multi-toggle switch-to="collapsed">
                <fo:external-graphic href="minus-icon.gif">
                </fo:external-graphic>
            </fo:multi-toggle>
            <fo:basic-link internal-destination="N28">2. Chapter
            </fo:basic-link>
        </fo:block>
        <fo:block start-indent="10mm">
            <fo:basic-link internal-destination="N35">2.1 Section
            </fo:basic-link>
        </fo:block>
        <fo:block start-indent="10mm">
            <fo:basic-link internal-destination="N43">2.2 Section
            </fo:basic-link>
        </fo:block>
    </fo:multi-case>
</fo:multi-switch>
<fo:block id="N4">1. Chapter
</fo:block>
<fo:block>Text
</fo:block>
<fo:block id="N11">1.1 Section
</fo:block>

```

```

<fo:block>Text
</fo:block>
<fo:block id="N19">1.2 Section
</fo:block>
<fo:block>Text
</fo:block>
<fo:block id="N28">2. Chapter
</fo:block>
<fo:block>Text
</fo:block>
<fo:block id="N35">2.1 Section
</fo:block>
<fo:block>Text
</fo:block>
<fo:block id="N43">2.2 Section
</fo:block>
<fo:block>Text
</fo:block>

```

6.9.1.1.2. Styling an XLink Based on the Active State

Input sample:

```

<p>Follow this <xlink:mylink xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:href="http://www.w3.org/TR"
    xlink:title="An Example"
    xlink:show="new"
    xlink:actuate="onRequest">link</xlink:mylink> to access all
TRs of the W3C.</p>

```

In this example an fo:basic-link contains a series of fo:multi-property-sets that specify various colors or text-decorations depending on the active state, and a wrapper around the fo:basic-link that allows for the merging of the properties of the fo:multi-properties with those of the appropriate fo:multi-property-sets.

XSL Stylesheet:

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    version='1.0'>
<xsl:template match="p">
    <fo:block>
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
<xsl:template match="xlink:mylink" xmlns:xlink="http://www.w3.org/1999/xlink">
    <xsl:variable name="show"><xsl:value-of select="@xlink:show"/>
</xsl:variable>
    <fo:multi-properties text-decoration="underline">
        <fo:multi-property-set active-state="link" color="blue"/>
        <fo:multi-property-set active-state="visited" color="red"/>
        <fo:multi-property-set active-state="active" color="green"/>
        <fo:multi-property-set active-state="hover" text-decoration="blink"/>
        <fo:multi-property-set active-state="focus" color="yellow"/>
        <fo:wrapper color="merge-property-values()">

```

```

        text-decoration="merge-property-values()">
    <fo:basic-link external-destination="http://www.w3.org/TR"
        show-destination="{ $show }">
        <xsl:attribute name="role">
            <xsl:value-of select="@xlink:title"/>
        </xsl:attribute>
        <xsl:apply-templates/>
    </fo:basic-link>
</fo:wrapper>
</fo:multi-properties>
</xsl:template>
</xsl:stylesheet>

```

Result Instance: elements and attributes in the fo: namespace

```

<fo:block>Follow this
  <fo:multi-properties text-decoration="underline">
    <fo:multi-property-set active-state="link" color="blue">
    </fo:multi-property-set>
    <fo:multi-property-set active-state="visited" color="red">
    </fo:multi-property-set>
    <fo:multi-property-set active-state="active" color="green">
    </fo:multi-property-set>
    <fo:multi-property-set active-state="hover" text-decoration="blink">
    </fo:multi-property-set>
    <fo:multi-property-set active-state="focus" color="yellow">
    </fo:multi-property-set>
    <fo:wrapper color="merge-property-values()"
      text-decoration="merge-property-values()">
      <fo:basic-link external-destination="http://www.w3.org/TR"
        show-destination="new" role="An Example">link
      </fo:basic-link>
    </fo:wrapper>
  </fo:multi-properties> to access all
  TRs of the W3C.
</fo:block>

```

6.9.2. fo:basic-link

Common Usage:

The fo:basic-link is used for representing the start resource of a simple one-directional single-target link. The object allows for traversal to the destination resource, typically by clicking on any of the containing areas.

Areas:

The fo:basic-link formatting object generates one or more *normal* inline-areas. The fo:basic-link returns these areas, any *page-level-out-of-line* areas, and any *reference-level-out-of-line* areas returned by the children of the fo:basic-link.

NOTE: An fo:simple-link can be enclosed in an fo:block to create a display area.

Constraints:

No area may have more than one normal child area returned by the same fo:basic-link formatting object.

The children of each normal area returned by an `fo:basic-link` must satisfy the constraints specified in § 4.7.3 – [Inline-building](#) on page 34.

Contents:

`(#PCDATA|%inline;|%block;)*`

In addition this formatting object may have a sequence of zero or more `fo:markers` as its initial children.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [Common Aural Properties](#) — § 7.5 on page 166
- [Common Border, Padding, and Background Properties](#) — § 7.6 on page 172
- [Common Margin Properties-Inline](#) — § 7.10 on page 210
- [“alignment-adjust”](#) — § 7.11.1 on page 221
- [“alignment-baseline”](#) — § 7.11.2 on page 223
- [“baseline-shift”](#) — § 7.11.3 on page 225
- [“destination-placement-offset”](#) — § 7.20.5 on page 279
- [“dominant-baseline”](#) — § 7.11.5 on page 227
- [“external-destination”](#) — § 7.20.6 on page 279
- [“id”](#) — § 7.26.2 on page 320
- [“indicate-destination”](#) — § 7.20.7 on page 280
- [“internal-destination”](#) — § 7.20.8 on page 280
- [“keep-together”](#) — § 7.17.3 on page 265
- [“keep-with-next”](#) — § 7.17.4 on page 266
- [“keep-with-previous”](#) — § 7.17.5 on page 267
- [“line-height”](#) — § 7.13.4 on page 241
- [“line-height-shift-adjustment”](#) — § 7.13.5 on page 243
- [“relative-position”](#) — § 7.18.4 on page 272
- [“show-destination”](#) — § 7.20.9 on page 280
- [“target-processing-context”](#) — § 7.20.13 on page 283
- [“target-presentation-context”](#) — § 7.20.12 on page 282
- [“target-stylesheet”](#) — § 7.20.14 on page 284

6.9.3. `fo:multi-switch`

Common Usage:

The `fo:multi-switch` wraps the specification of alternative sub-trees of formatting objects (each sub-tree being within an `fo:multi-case`), and controls the switching (activated via `fo:multi-toggle`) from one alternative to another.

The direct children of an `fo:multi-switch` object are `fo:multi-case` objects. Only a single `fo:multi-case` may be visible at a single time. The user may switch between the available multi-cases.

Each `fo:multi-case` may contain one or more `fo:multi-toggle` objects, which controls the `fo:multi-case` switching of the `fo:multi-switch`.

NOTE: An `fo:multi-switch` can be used for many interactive tasks, such as table-of-content views, embedding link targets, or generalized (even multi-layered hierarchical), next/previous views. The latter are today normally handled in HTML by next/previous links to other documents, forcing the whole document to be replaced whenever the users decides to move on.

Areas:

The fo:multi-switch formatting object does not generate any areas. The fo:multi-switch formatting object returns the sequence of areas returned by the currently visible fo:multi-case. If there is no currently visible fo:multi-case no areas are returned.

Trait Derivation:

The *currently-visible-multi-case* trait has as its initial value a reference to the first fo:multi-case child that has a value of "show" of the *starting-state* trait. If there is no such child, it has a value indicating that there is no currently visible fo:multi-case. When an fo:multi-toggle is actuated, its closest ancestral fo:multi-switch's *currently-visible-multi-case* trait value changes to refer to the fo:multi-case selected by the "switch-to" property value of the fo:multi-toggle. Once the *currently-visible-multi-case* trait gets a value indicating that there is no currently visible fo:multi-case, it becomes impossible to actuate an fo:multi-toggle in this fo:multi-switch.

Constraints:

The order of the sequence of areas returned by the fo:multi-switch is the same as the order of the areas returned by the currently visible fo:multi-case.

Contents:

([multi-case](#)*)

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- [“auto-restore”](#) — § 7.20.2 on page 277
- [“id”](#) — § 7.26.2 on page 320

6.9.4. fo:multi-case

Common Usage:

The fo:multi-case is used to contain (within an fo:multi-switch) each alternative sub-tree of formatting objects among which the parent fo:multi-switch will choose one to show and will hide the rest.

Areas:

The fo:multi-case formatting object does not generate any areas. The fo:multi-case formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:multi-case.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:multi-case is the same order as the children are ordered under the fo:multi-case.

Contents:

(#PCDATA|[%inline;](#)|[%block;](#))*

An fo:multi-case is only permitted to have children that would be permitted to be children of the parent of the fo:multi-switch that is the parent of the fo:multi-case, except that an fo:multi-case may not contain fo:marker children. In particular, it can contain fo:multi-toggle objects (at any depth), which controls the fo:multi-case switching.

This restriction applies recursively.

NOTE: For example, an fo:multi-case whose parent fo:multi-switch is a child of another fo:multi-case may only have children that would be permitted in place of the outer fo:multi-case's parent fo:multi-switch.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- “id” — § 7.26.2 on page 320
- “starting-state” — § 7.20.10 on page 281
- “case-name” — § 7.20.3 on page 278
- “case-title” — § 7.20.4 on page 278

6.9.5. fo:multi-toggle

Common Usage:

The fo:multi-toggle is typically used to establish an area that when actuated (for example implemented as "clicked"), has the effect of switching from one fo:multi-case to another. The "switch-to" property value of the fo:multi-toggle typically matches the "case-name" property value of the fo:multi-case to switch to.

Areas:

The fo:multi-toggle formatting object does not generate any areas. The fo:multi-toggle formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:multi-toggle. Each of the areas returned by the fo:multi-toggle has a *switch-to* trait with the same value as on the returning fo:multi-toggle.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:multi-toggle is the same order as the children are ordered under the fo:multi-toggle.

Activating an area returned by an fo:multi-toggle causes a change to the value of the *currently-visible-multi-case* of the closest ancestor fo:multi-switch. (See § 7.20.11 – “switch-to” on page 281 for how the *switch-to* value selects an fo:multi-case.)

Contents:

```
(#PCDATA|%inline;|%block;)*
```

An fo:multi-toggle is only permitted as a descendant of an fo:multi-case.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- “id” — § 7.26.2 on page 320
- “switch-to” — § 7.20.11 on page 281

6.9.6. fo:multi-properties

Common Usage:

The fo:multi-properties is used to switch between two or more property sets that are associated with a given portion of content.

NOTE: An fo:multi-properties formatting object can be used to give different appearances to a given portion of content. For example, when a link changes from the not-yet-visited state to the visited-state, this could change the set of properties that would be used to format the content. Designers should be careful in choosing which

properties they change, because many property changes could cause reflowing of the text which may not be desired in many circumstances. Changing properties such as "color" or "text-decoration" should not require re-flowing the text.

The direct children of an fo:multi-properties formatting object is an ordered set of fo:multi-property-set formatting objects followed by a single fo:wrapper formatting object. The properties, specified on the fo:wrapper, that have been specified with a value of "merge-property-values()" will take a value that is a merger of the value on the fo:multi-properties and the specified values on the fo:multi-property-set formatting objects that apply.

Areas:

The fo:multi-properties formatting object does not generate any areas. The fo:multi-properties formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:multi-properties.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:multi-properties is the same order as the children are ordered under the fo:multi-properties.

Contents:

(multi-property-set+, wrapper)

The properties that should take a merged value shall be specified with a value of "merge-property-values()". This function, when applied on an fo:wrapper that is a direct child of an fo:multi-properties, merges the applicable property definitions on the fo:multi-property-set siblings.

The following properties apply to this formatting object:

- [Common Accessibility Properties](#) — § 7.3 on page 162
- “id” — § 7.26.2 on page 320

6.9.7. fo:multi-property-set

Common Usage:

The fo:multi-property-set auxiliary formatting object is used to specify an alternative set of formatting properties that can be used to provide an alternate presentation of the children flow objects of the fo:wrapper child of the parent of this fo:multi-property-set.

Areas:

The fo:multi-property-set formatting object does not generate or return any areas. It simply holds a set of traits that may be accessed by expressions.

Constraints:

None.

Contents:

EMPTY

The following properties apply to this formatting object:

- “id” — § 7.26.2 on page 320
- “active-state” — § 7.20.1 on page 276

6.10. Out-of-Line Formatting Objects

6.10.1. Introduction

6.10.1.1. Floats

The fo:float formatting object is used for two distinct purposes. First, so that during the normal placement of content, some related content is formatted into a separate area at the beginning of a page where it is available to be read without immediately intruding on the reader. The areas generated by this kind of fo:float are called before-floats. An fo:float is specified to generate before-floats if it has a "float" property value of "before". The constraints on placing before-floats on a page are described in the § 6.10.1.3 – Conditional Sub-Regions on page 151 section of this introduction and in the description of the fo:float formatting object.

Second, the fo:float formatting object is used when an area is intended to float to one side, with normal content flowing alongside the floated area. The areas generated by this kind of fo:float are called side-floats. A side-float is always made a child of the nearest ancestor reference-area. The edge of the reference-area towards which the side-float floats is controlled by the value of the "float" property.

Flowing normal content flowing alongside side-floats is realized by increasing the start-intrusion-adjustment or the end-intrusion-adjustment of normal child areas of the parent reference-area of the side-float.

The "clear" property applies to any block-level formatting object. If the value of this property for a particular formatting object is any value other than "none", then the areas generated by the block will be positioned to ensure that their border-rectangles do not overlap the allocation-rectangles of the applicable side-floats as determined by the "clear" property value.

6.10.1.2. Footnotes

The fo:footnote formatting object is used to generate both a footnote and its citation. The fo:footnote has two children, which are both required to be present. The first child is an fo:inline formatting object, which is formatted to produce the footnote citation. The second child is an fo:footnote-body formatting object which generates the content (or body) of the footnote.

The actual areas generated by the descendants of the fo:footnote-body formatting object are determined by the formatting objects that comprise the descendant subtree. For example, the footnote could be formatted with a label and an indented body by using the fo:list-block formatting object within the fo:footnote-body.

6.10.1.3. Conditional Sub-Regions

The region-body has two conditional sub-regions which implicitly specify corresponding reference-areas called *before-float-reference-area* and *footnote-reference-area*. These reference-areas are conditionally generated as children of the region-reference-area. The before-float-reference-area is generated only if the page contains one or more areas with area-class "xsl-before-float". The footnote-reference-area is generated only if the page contains one or more areas with area-class "xsl-footnote".

The conditionally generated areas borrow space in the *block-progression-dimension* (this is "height" when the writing-mode is "lr-tb") within the region-reference-area, at the expense of the main-reference-area. Whether or not a conditionally generated area is actually generated depends, additionally, on whether there is sufficient space left in the main-reference-area.

There may be limits on how much space conditionally generated areas can borrow from the region-reference-area. It is left to the user agent to decide these limits.

The block-progression-dimension of the main-reference-area is set equal to the block-progression-dimension of the allocation-rectangle of the region-reference-area minus the sum of the sizes in the block-progression-direction of the allocation-rectangles of the conditionally generated reference-areas that were actually generated. The main-reference-area is positioned to immediately follow the after-edge of the allocation-rectangle of the before-float-reference-area. This positions the after-edge of the main-reference-area to coincide with the before-edge of the allocation-rectangle of the footnote-reference-area. In addition to the constraints normally determined by the region-reference-area, the *inline-progression-dimension* (this is "width" when the writing-mode is "lr-tb") of a conditionally generated reference-area is constrained to match the inline-progression-dimension of the main-reference-area.

Each conditionally generated reference-area may additionally contain a sequence of areas used to separate the reference-area from the main-reference-area. The sequence of areas is the sequence returned by formatting a fo:static-content specified in the page-sequence that is being used to format the page.

If there is a fo:static-content in a page-sequence whose "region-name" property value is "xsl-before-float-separator", then the areas returned by formatting the fo:static-content are inserted in the proper order as the last children of a before-float-reference-area that is generated using the same page-master, provided the main-reference-area on the page is not empty.

If there is an fo:static-content whose "region-name" property value is "xsl-footnote-separator", then the areas returned by formatting the fo:static-content are inserted in the proper order as the initial children of a footnote-reference-area that is generated using the same page-master.

An interactive user agent may choose to create "hot links" to the footnotes from the footnote-citation, or create "hot links" to the before-floats from an implicit citation, instead of realizing conditional sub-regions.

The generation of areas with area-class "xsl-before-float" or "xsl-footnote" is specified in the descriptions of the formatting objects that initially return areas with those area-classes.

6.10.2. fo:float

Common Usage:

The fo:float formatting object is typically used either to cause an image to be positioned in a separate area at the beginning of a page, or to cause an image to be positioned to one side, with normal content flowing around and along-side the image.

Areas:

The fo:float generates a single normal inline-area with no children, and one or more block-areas that all share the same area-class, which is either "xsl-before-float", "xsl-side-float" or "xsl-normal" as specified by the "float" property. (An fo:float generates normal block-areas if its "float" property value is "none".)

Areas with area-class "xsl-side-float" are reference areas.

An area with area-class "xsl-before-float" is placed as a child of a before-float-reference-area.

Constraints:

The normal inline-area generated by the fo:float shall be placed in the area tree as though the fo:float

had a "keep-with-previous" property with value "always". The inline-area has a length of zero for both the inline-progression-dimension and block-progression-dimension.

The term *anchor-area* is used here to mean the normal inline-area generated by the fo:float.

No area may have more than one child block-area returned by the same fo:float formatting object.

Areas with area-class "xsl-before-float" must be properly ordered within the area tree relative to other areas with the same area-class.

The padding-, border-, and content-rectangles of the block-areas generated by fo:float all coincide. That is, the padding and border are zero at all edges of the area.

The following constraints apply to fo:float formatting objects that generate areas with area-class "xsl-before-float":

- It is an error if the fo:float occurs as a descendant of a flow that is not assigned to a region-body, or of an fo:block-container that generates absolutely positioned areas. In either case, the fo:float shall be formatted as though its "float" property was "none".
- A block-area generated by the fo:float may only be descendant from a before-float-reference-area that is (a) descendant from a "region-reference-area" generated using the region-master for the region to which the flow that has the fo:float as a descendant is assigned, and (b) is descendant from the same page containing the anchor-area, or from a page following that page.
- The fo:float may not generate any additional block-areas unless the page containing the preceding block-area generated by the fo:float contains no other areas with area-class "xsl-before-float", has an empty main-reference-area, and does not contain a footnote-reference-area.
- The "clear" property does not apply.

The following constraints apply to fo:float formatting objects that generate areas with area-class "xsl-side-float":

- Each block-area generated by the fo:float is placed either as a child of the nearest ancestor reference-area of the anchor-area or as a child of a later reference-area in the same reference-area chain.
- A block-area generated by one fo:float may not be placed so that it overlaps another area with area-class "xsl-side-float".
- The description in section 9.5 of [CSS2] shall be used to determine the formatting of the fo:float and the rendering of normal line-areas that overlap a block-area generated by the fo:float, with these modifications:
 - All references to left and right shall be interpreted as their corresponding writing-mode relative directions "start" and "end". The "float" property will additionally have the writing-mode relative values "start" and "end".
 - All references to top and bottom shall be interpreted as their corresponding writing-mode relative directions "before" and "after".
 - The phrase "current line box" shall be interpreted to mean the nearest ancestor reference-area of the anchor-area, or a later reference-area in the same reference-area chain.
 - The areas generated by the fo:float derive their length in the inline-progression-dimension intrinsically from their child areas; the length is not determined by an explicit property value.

- The length of a block-area generated by the fo:float, in the inline-progression-direction, increases either the start-intrusion-adjustment or the end-intrusion-adjustment, depending on which side the block-area floats to, of each normal child area of the parent reference-area of the generated block-area, provide the border-rectangle of the child area overlaps the floated area in the inline-progression-direction.

Contents:

(%block;)+

An fo:float is not permitted to have an fo:float, fo:footnote or fo:marker as a descendant.

Additionally, an fo:float is not permitted to have as a descendant an fo:block-container that generates an absolutely positioned area.

The following properties apply to this formatting object:

- “float” — § 7.16.2 on page 262
- “clear” — § 7.16.1 on page 259

6.10.3. fo:footnote

Common Usage:

The fo:footnote is typically used to produce footnote-citations within the region-body of a page and the corresponding footnote in a separate area nearer the after-edge of the page.

Areas:

The fo:footnote formatting object does not generate any areas. The fo:footnote formatting object returns the areas generated and returned by its child fo:inline formatting object.

Additionally the fo:footnote formatting object returns the block-areas with area class "xsl-footnote" generated by its fo:footnote-body child. An area with area-class "xsl-footnote" is placed as a child of a footnote-reference-area.

Constraints:

The term *anchor-area* is defined to mean the last area that is generated and returned by the fo:inline child of the fo:footnote.

A block-area returned by the fo:footnote is only permitted as a descendant from a footnote-reference-area that is (a) descendant from a "region-reference-area" generated using the region-master for the region to which the flow that has the fo:footnote as a descendant is assigned, and (b) is descendant from the same page that contains the anchor-area, or from a page following the page that contains the anchor-area.

The second block-area and any additional block-areas returned by an fo:footnote must be placed on the immediately subsequent pages to the page containing the first block-area returned by the fo:footnote, before any other content is placed. If a subsequent page does not contain a region-body, the user agent must use the region-master of the last page that did contain a region-body to hold the additional block-areas.

It is an error if the fo:footnote occurs as a descendant of a flow that is not assigned to a region-body, or of an fo:block-container that generates absolutely positioned areas. In either case, the block-areas generated by the fo:footnote-body child of the fo:footnote shall be returned to the parent of the fo:footnote and placed in the area tree as though they were normal block-level areas.

Contents:`(inline, footnote-body)`

An fo:footnote is not permitted to have an fo:float, fo:footnote, or fo:marker as a descendant.

Additionally, an fo:footnote is not permitted to have as a descendant an fo:block-container that generates an absolutely positioned area.

6.10.4. fo:footnote-body*Common Usage:*

The fo:footnote-body is used to generate the footnote content.

Areas:

The fo:footnote-body generates and returns one or more block-level areas with area-class "xsl-footnote".

Constraints:

The fo:footnote-body is only permitted as a child of an fo:footnote.

No area may have more than one child block-area returned by the same fo:footnote-body formatting object.

Areas with area-class "xsl-footnote" must be properly ordered within the area tree relative to other areas with the same area-class.

Contents:`(%block;)+`**6.11. Other Formatting Objects****6.11.1. Introduction**

The following example shows the use of the fo:wrapper formatting object that has no semantics but acts as a "carrier" for inherited properties.

6.11.1.1. Example

Input sample:

```
<doc>
<p>This is an <emph>important word</emph> in this
sentence that also refers to a <code>variable</code>.</p>
</doc>
```

The "emph" elements are to be presented using a bold font and the "code" elements are using a Courier font.

XSL Stylesheet:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>
<xsl:template match="p">
  <fo:block>
```

```

    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="emph">
  <fo:wrapper font-weight="bold">
    <xsl:apply-templates/>
  </fo:wrapper>
</xsl:template>
<xsl:template match="code">
  <fo:wrapper font-family="Courier">
    <xsl:apply-templates/>
  </fo:wrapper>
</xsl:template>
</xsl:stylesheet>

```

fo: element and attribute tree:

```

<fo:block xmlns:fo="http://www.w3.org/1999/XSL/Format">This is an
<fo:wrapper font-weight="bold">important word</fo:wrapper>
in this sentence that also refers to a
<fo:wrapper font-family="Courier">variable</fo:wrapper>.
</fo:block>

```

6.11.2. fo:wrapper

Common Usage:

The fo:wrapper formatting object is used to specify inherited properties for a group of formatting objects.

Areas:

The fo:wrapper formatting object does not generate any areas. The fo:wrapper formatting object returns the sequence of areas created by concatenating the sequences of areas returned by each of the children of the fo:wrapper.

Trait Derivation:

Except for "id", the fo:wrapper has no properties that are directly used by it. However, it does serve as a carrier to hold inheritable properties that are utilized by its children.

Constraints:

The order of concatenation of the sequences of areas returned by the children of the fo:wrapper is the same order as the children are ordered under the fo:wrapper.

Contents:

```
(#PCDATA|%inline;|%block;)*
```

An fo:wrapper is only permitted to have children that would be permitted to be children of the parent of the fo:wrapper, with two exceptions:

- An fo:wrapper may always have a sequence of zero or more fo:markers as its initial children.
- An fo:wrapper that is a child of an fo:multi-properties is only permitted to have children that would be permitted in place of the fo:multi-properties.

This restriction applies recursively.

NOTE: For example an fo:wrapper that is a child of another fo:wrapper may only have children that would be permitted to be children of the parent fo:wrapper.

The following properties apply to this formatting object:

- “id” — § 7.26.2 on page 320

6.11.3. fo:marker

Common Usage:

The fo:marker is used in conjunction with fo:retrieve-marker to produce running headers or footers. Typical examples include:

- dictionary headers showing the first and last word defined on the page.
- headers showing the page's chapter and section titles.

The fo:marker has to be an initial child of its parent formatting object.

Areas:

The fo:marker does not directly produce any area. Its children may be retrieved and formatted from within an fo:static-content, using an fo:retrieve-marker whose "retrieve-class-name" property value is the same as the "marker-class-name" property value of this fo:marker.

Constraints:

An fo:marker is only permitted as the descendant of an fo:flow.

Note: Property values set on an fo:marker or its ancestors will not be inherited by the children of the fo:marker when they are retrieved by an fo:retrieve-marker.

It is an error if two or more fo:markers that share the same parent have the same "marker-class-name" property value.

Contents:

(#PCDATA | %inline; | %block;) *

An fo:marker may contain any formatting objects that are permitted as a replacement of any fo:retrieve-marker that retrieves the fo:marker's children.

The following properties apply to this formatting object:

- “marker-class-name” — § 7.21.1 on page 284

6.11.4. fo:retrieve-marker

Common Usage:

The fo:retrieve-marker is used in conjunction with fo:marker to produce running headers or footers. Typical examples include:

- dictionary headers showing the first and last word defined on the page.
- headers showing the page's chapter and section titles.

Areas:

The fo:retrieve-marker does not directly generate any area. It is (conceptually) replaced by the children of the fo:marker that it retrieves.

Trait Derivation:

The properties and traits specified on the ancestors of the `fo:retrieve-marker` are taken into account when formatting the children of the retrieved `fo:marker` as if the children had the same ancestors as the `fo:retrieve-marker`.

Constraints:

An `fo:retrieve-marker` is only permitted as the descendant of an `fo:static-content`.

The `fo:retrieve-marker` specifies that the children of a selected `fo:marker` shall be formatted as though they replaced the `fo:retrieve-marker` in the formatting tree.

The properties of the `fo:retrieve-marker` impose a hierarchy of preference on the areas of the area tree. Each `fo:marker` is conceptually attached to each normal area returned by the `fo:marker's` parent formatting object. Additionally, an `fo:marker` is conceptually attached to each non-normal area that is directly generated by the `fo:marker's` parent formatting object. Conversely, areas generated by any descendant of an `fo:flow` may have zero or more `fo:marker's` conceptually attached. The `fo:marker` whose children are retrieved is the one that is (conceptually) attached to the area that is at the top of this hierarchy.

Every area in the hierarchy is considered preferential to, or "better" than, any area below it in the hierarchy. When comparing two areas to determine which one is better, the terms "first" and "last" refer to the pre-order traversal order of the area tree.

The term "containing page" is used here to mean the page that contains the first area generated or returned by the children of the retrieved `fo:marker`.

An area that has an attached `fo:marker` whose "marker-class-name" property value is the same as the "retrieve-class-name" property value of the `fo:retrieve-marker` is defined to be a qualifying area. Only qualifying areas have positions in the hierarchy.

A qualifying area within a page is better than any qualifying area within a preceding page, except that areas do not have a position in the hierarchy if they are within pages that follow the containing page. If the "retrieve-boundary" property has a value of "page-sequence", then an area does not have a position in the hierarchy if it is on a page from a page-sequence preceding the page-sequence of the containing page. If the "retrieve-boundary" property has a value of "page", then an area does not have a position in the hierarchy if it is not on the containing page.

If the value of the "retrieve-position" property is "first-starting-within-page", then the first qualifying area in the containing page whose "is-first" trait has a value of "true" is better than any other area. If there is no such area, then the first qualifying area in the containing page is better than any other area.

If the value of the "retrieve-position" property is "first-including-carryover", then the first qualifying area in the containing page is better than any other area.

If the value of the "retrieve-position" property is "last-starting-within-page", then the last qualifying area in the containing page whose "is-first" trait has a value of "true" is better than any other area. If there is no such area, then the last qualifying area in the containing page is better than any other area.

If the value of the "retrieve-position" property is "last-ending-within-page", then the last qualifying area in the containing page whose "is-last" trait has a value of "true" is better than any other area. If there is no such area, then the last qualifying area in the containing page is better than any other area.

If the hierarchy of areas is empty, no formatting objects are retrieved.

Contents:

EMPTY

The following properties apply to this formatting object:

- “[retrieve-class-name](#)” — § 7.21.2 on page 285
- “[retrieve-position](#)” — § 7.21.3 on page 285
- “[retrieve-boundary](#)” — § 7.21.4 on page 286

7. Formatting Properties

7.1. Description of Property Groups

The following sections describe the properties of the XSL formatting objects.

A number of properties are copied from the CSS2 specification. In addition, the CSS2 errata all apply. See [\[CSS2\]](#).

- The first eight sets of property definitions have been arranged into groups based on similar functionality and the fact that they apply to many formatting objects. In the formatting object descriptions the group name is referred to rather than referring to the individual properties.
 - Common Absolute Position Properties

This set of properties controls the position and size of formatted areas with absolute positioning.
 - Common Aural Properties

This group of properties controls the aural rendition of the content of a formatting object. They appear on all formatting objects that contain content and other formatting objects that group other formatting objects and where that grouping is necessary for the understanding of the aural rendition. An example of the latter is `fo:table-and-caption`.
 - Common Border, Padding, and Background Properties

This set of properties controls the backgrounds and borders on the block-areas and inline-areas.
 - Common Font Properties

This set of properties controls the font selection on all formatting objects that can contain text.
 - Common Hyphenation Properties

Control of hyphenation for line-breaking, including language, script, and country.
 - Common Margin Properties-Block

These properties set the spacing and indents surrounding block-level formatting objects.
 - Common Margin Properties-Inline

These properties set the spacing surrounding inline-level formatting objects.
- The remaining properties are used on a number of formatting objects. These are arranged into clusters of similar functionality to organize the property descriptions. In the formatting object description the individual properties are referenced.
 - Area Alignment Properties

Properties that control the alignment of inline-areas with respect to each other, particularly in relation to the mixing of different baselines for different scripts. In addition, there are two properties: "display-align" and "relative-align" that control the placement of block-areas.

- Area Dimension Properties

Properties that control the dimensions of both block-areas and inline-areas.

- Block and Line-related Properties

Properties that govern the construction of line-areas and the placement of these line-areas within containing block-areas.

- Character Properties

Properties that govern the presentation of text, including word-spacing, letter-spacing, and word-space treatment and suppression.

- Color-related Properties

Properties that govern color and color-model selection.

- Float-related properties

Properties governing the placement of both side-floats (start- and end-floats) and before-floats ("top" floats in "lr-tb" writing-mode).

- Keeps and Breaks Properties

Properties that control keeps and breaks across pages, columns, and lines, including widow and orphan control and keeping content together.

- Layout-related Properties

These properties control what is "top" ("reference-orientation") as well as clipping, overflow, and column-spanning conditions.

- Leader and Rule Properties

Properties governing the construction of leaders and horizontal rules.

- Properties for Links

Properties governing the presentation and actions associated with links.

- Properties for Markers

Properties governing the creation and retrieval of markers. Markers are used typically for "dictionary" headers and footers.

- Properties for Number to String Conversions

Properties used in the construction of page-numbers and other formatter-based numbering.

- Pagination and Layout Properties

These properties govern the sequencing, layout, and instantiation of pages, including: the page size and orientation, sizes of regions on the page-master, the identification and selection of page-masters, division of the body region into columns, and the assignment of content flows to layout regions.

- Table Properties

Properties governing the layout and presentation of tables.

- Writing-mode-related Properties

Properties related to various aspects of "directionality" and writing-mode influencing block-progression-direction and inline-progression-direction.

- Miscellaneous Properties

These properties did not reasonably fit into any of the other categories.

- Shorthand Properties

Shorthand properties that are part of the complete conformance set. Shorthands expand to the individual properties that may be used in place of shorthands.

7.2. XSL Areas and the CSS Box Model

This section describes how to interpret property descriptions which incorporate the CSS2 definition of the same property. In CSS2, "boxes" are generated by "elements" in the same way that XSL areas are generated by formatting objects. Any references in the CSS2 definition to "boxes" are to be taken as referring to "areas" in the XSL area model, and where "element" appears in a CSS2 definition it should be taken to refer to a "formatting object".

The position and size of a box are normally taken to refer to the position and size of the area's content-rectangle. Additional correspondences between the CSS2 Box Model and the XSL Area Model are contained in the following table.

Box	Area
top content edge	top edge of the content-rectangle
padding edge	padding-rectangle
content area	interior of the content-rectangle
padding area	region between the content-rectangle and the padding-rectangle
border area	region between the padding-rectangle and the border-rectangle
background	background
containing block	closest ancestor block-area
caption	area generated by fo:table-caption
inline box	inline-area
line box	line-area
block box	block-area which is not a line-area
page box	page-area

Box margins map to area traits in accordance with the description of how area traits are computed from property values in [§ 5 – Property Refinement / Resolution](#) on page 39.

7.3. Common Accessibility Properties

7.3.1. “source-document”

XSL Definition:

<i>Value:</i>	<uri-specification> [<uri-specification>]* none inherit
<i>Initial:</i>	none
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	all

It is used by all formatting objects that can be contained in fo:flow or fo:static-content (all formatting objects that can be directly created from an XML source element).

Values have the following meanings:

none

The source document is transient, unknown, or unspecified.

<uri-specification>

A URI-specification giving a reference to the (sub)resource used as input to the stylesheet.

This property provides a pointer back to the original XML document(s) used to create this formatting object tree, in accordance with the Dublin Core definition of "Source" ("A Reference to a resource from which the present resource is derived." See: <http://purl.org/DC/documents/rec-dces-19990702.htm>.) The value is not validated by and has no inherent standardized semantics for any XSL processor.

W3C Accessibility guidelines strongly encourage the use of this property either on the fo:root or on the first formatting object generated from a given source document.

The URI reference is useful for alternate renderers (aural readers, etc.) whenever the structure of the formatting object tree is inappropriate for that renderer.

7.3.2. “role”

XSL Definition:

<i>Value:</i>	<string> <uri-specification> none inherit
<i>Initial:</i>	none
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	all

It is used by all formatting objects that can be contained in fo:flow or fo:static-content (all formatting objects that can be directly created from an XML source element).

Values have the following meanings:

none

Indicates that no semantic tag is cited by this formatting object.

<string>

The value is a string representing a semantic identifier that may be used in rendering this formatting object.

<uri-specification>

An URI-specification, indicating an RDF resource [RDF]; that is, an XML object that is syntactically valid with respect to the RDF grammar.

This property provides a hint for alternate renderers (aural readers, etc.) as to the role of the XML element or elements that were used to construct this formatting object, if one could be identified during XSLT tree construction. This information can be used to prepare alternate renderings when the normal rendering of a formatting object is not appropriate or satisfactory; for example, the role information can be used to provide better aural renderings of visually formatted material.

To aid alternate renderers, the *<string>* value should be the qualified name (QName [W3C XML Names]) of the element from which this formatting object is constructed. If a QName does not provide sufficient context, the *<uri-specification>* can be used to identify an RDF resource that describes the role in more detail. This RDF resource may be embedded in the result tree and referenced with a relative URI or fragment identifier, or the RDF resource may be external to the result tree. This specification does not define any standard QName or RDF vocabularies; these are frequently application area dependent. Other groups, for example the Dublin Core, have defined such vocabularies.

This property is not inherited, but all subsidiary nodes of this formatting object that do not bear a role property should utilize the same alternate presentation properties. (It is not inherited because knowledge of the start and end of the formatting object subtree generated by the element may be needed by the renderer.)

7.4. Common Absolute Position Properties

7.4.1. “absolute-position”

A Property Derived from a CSS2 Property.

<i>Value:</i>	auto absolute fixed inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:block-container
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

There is no absolute-positioning constraint. Positioning is in accordance with the relative-position property.

absolute

The area's position (and possibly size) is specified with the "left", "right", "top", and "bottom" properties. These properties specify offsets with respect to the area's containing area. Absolutely positioned areas are taken out of the normal flow. This means they have no impact on the layout of later siblings. Also, though absolutely positioned areas have margins, they do not collapse with any other margins.

fixed

The area's position is calculated according to the "absolute" model, but in addition, the area is fixed with respect to some reference. In the case of continuous media, the area is fixed with respect to the viewport (and doesn't move when scrolled). In the case of paged media, the area is fixed with respect to the page, even if that page is seen through a viewport (in the case of a print-preview, for example). Authors may wish to specify "fixed" in a media-dependent way. For instance, an author may want an area to remain at the top of the viewport on the screen, but not at the top of each printed page.

The following additional restrictions apply for paged presentations:

- Only objects with absolute-position="auto" may have page/column breaks.
For other values any keep and break properties are ignored.
- The area generated is a descendant of the page-area where the first area from the object would have been placed had the object had absolute-position="auto" specified.

7.4.2. "top"

CSS2 Definition:

Value: <length> | <percentage> | auto | inherit

Initial: auto

Applies to: positioned elements

Inherited: no

Percentages: refer to height of containing block

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-top>.

The "top" property specifies how far a box's top content edge is offset below the top edge of the box's containing block.

XSL modifications to the CSS definition:

See definition of property left (§ 7.4.5 – “left” on page 165).

7.4.3. “right”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> auto inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	positioned elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to height of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-right>.

The "right" property specifies how far a box's right content edge is offset to the left of the right edge of the box's containing block.

XSL modifications to the CSS definition:

See definition of property left (§ 7.4.5 – “left” on page 165).

7.4.4. “bottom”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> auto inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	positioned elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to height of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-bottom>.

The "bottom" property specifies how far a box's bottom content edge is offset above the bottom of the box's containing block.

XSL modifications to the CSS definition:

See definition of property left (§ 7.4.5 – “left” on page 165).

7.4.5. “left”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> auto inherit
---------------	--

<i>Initial:</i>	auto
<i>Applies to:</i>	positioned elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to height of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-left>.

The "left" property specifies how far a box's left content edge is offset to the right of the left edge of the box's containing block.

The values of the four (position offset) properties have the following meanings:

auto

The effect of this value depends on which of related properties have the value "auto" as well. See the sections on the width and height of absolutely positioned, non-replaced elements for details.

<length>

The offset is a fixed distance from the reference edge.

<percentage>

The offset is a percentage of the containing block's width (for "left" or "right") or "height" (for "top" and "bottom"). For "top" and "bottom", if the "height" of the containing block is not specified explicitly (i.e., it depends on content height), the percentage value is interpreted like "auto".

For absolutely positioned boxes, the offsets are with respect to the box's containing block. For relatively positioned boxes, the offsets are with respect to the outer edges of the box itself (i.e., the box is given a position in the normal flow, then offset from that position according to these properties).

XSL modifications to the CSS definition:

These properties set the position of the content-rectangle of the associated area.

If both "top" and "bottom" are specified, the height of the content-rectangle is overridden. If both "left" and "right" are specified, the width of the content-rectangle is overridden.

7.5. Common Aural Properties

7.5.1. “azimuth”

CSS2 Definition:

<i>Value:</i>	<angle> [[left-side far-left left center-left center center-right right far-right right-side] behind] leftwards rightwards inherit
<i>Initial:</i>	center

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-azimuth>.

7.5.2. “cue-after”

CSS2 Definition:

Value: <uri-specification> | none | inherit

Initial: none

Applies to: all elements

Inherited: no

Percentages: N/A

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-cue-after>.

XSL modifications to the CSS definition:

The <uri> value has been changed to a <uri-specification>.

7.5.3. “cue-before”

CSS2 Definition:

Value: <uri-specification> | none | inherit

Initial: none

Applies to: all elements

Inherited: no

Percentages: N/A

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-cue-before>.

XSL modifications to the CSS definition:

The <uri> value has been changed to a <uri-specification>.

7.5.4. “elevation”

CSS2 Definition:

Value: <angle> | below | level | above | higher | lower | inherit

<i>Initial:</i>	level
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-elevation>.

7.5.5. “pause-after”

CSS2 Definition:

<i>Value:</i>	<time> <percentage> inherit
<i>Initial:</i>	depends on user agent
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	see prose
<i>Media:</i>	aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-pause-after>.

7.5.6. “pause-before”

CSS2 Definition:

<i>Value:</i>	<time> <percentage> inherit
<i>Initial:</i>	depends on user agent
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	see prose
<i>Media:</i>	aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-pause-before>.

7.5.7. “pitch”

CSS2 Definition:

<i>Value:</i>	<frequency> x-low low medium high x-high inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes

Percentages: N/A
Media: aural
 CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-pitch>.

7.5.8. “pitch-range”

CSS2 Definition:

Value: <number> | inherit
Initial: 50
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: aural
 CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-pitch-range>.

7.5.9. “play-during”

CSS2 Definition:

Value: <uri-specification> mix? repeat? | auto | none | inherit
Initial: auto
Applies to: all elements
Inherited: no
Percentages: N/A
Media: aural
 CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-play-during>.

XSL modifications to the CSS definition:

The <uri> value has been changed to a <uri-specification>.

7.5.10. “richness”

CSS2 Definition:

Value: <number> | inherit
Initial: 50
Applies to: all elements
Inherited: yes
Percentages: N/A

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-richness>.

7.5.11. “speak”

CSS2 Definition:

Value: normal | none | spell-out | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-speak>.

7.5.12. “speak-header”

CSS2 Definition:

Value: once | always | inherit

Initial: once

Applies to: elements that have table header information

Inherited: yes

Percentages: N/A

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/tables.html#propdef-speak-header>.

7.5.13. “speak-numeral”

CSS2 Definition:

Value: digits | continuous | inherit

Initial: continuous

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-speak-numeral>.

7.5.14. “speak-punctuation”

CSS2 Definition:

<i>Value:</i>	code none inherit
<i>Initial:</i>	none
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-speak-punctuation>.

7.5.15. “speech-rate”

CSS2 Definition:

<i>Value:</i>	<number> x-slow slow medium fast x-fast faster slower inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-speech-rate>.

7.5.16. “stress”

CSS2 Definition:

<i>Value:</i>	<number> inherit
<i>Initial:</i>	50
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-stress>.

7.5.17. “voice-family”

CSS2 Definition:

<i>Value:</i>	[[<specific-voice> <generic-voice>],]* [<specific-voice>
---------------	---

<generic-voice>] | inherit

Initial: depends on user agent

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-voice-family>.

7.5.18. “volume”

CSS2 Definition:

Value: <number> | <percentage> | silent | x-soft | soft | medium | loud | x-loud | inherit

Initial: medium

Applies to: all elements

Inherited: yes

Percentages: refer to inherited value

Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-volume>.

7.6. Common Border, Padding, and Background Properties

The following common-border-padding-and-background-properties are taken from CSS2. Those "border", "padding", and "background" properties that have a before, after, start, or end suffix are writing-mode relative and are XSL-only properties.

7.6.1. “background-attachment”

CSS2 Definition:

Value: scroll | fixed | inherit

Initial: scroll

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/colors.html#propdef-background-attachment>.

scroll

The background-image may scroll with the enclosing object.

fixed

The background-image is to be fixed within the viewable area of the enclosing object.

If a background-image is specified, this property specifies whether it is fixed with regard to the viewport (fixed) or scrolls along with the document (scroll).

Even if the image is fixed, it is still only visible when it is in the background or padding area of the element. Thus, unless the image is tiled ("background-repeat: repeat"), it may be invisible.

User agents may treat fixed as scroll. However, it is recommended they interpret fixed correctly, at least for the HTML and BODY elements, since there is no way for an author to provide an image only for those browsers that support fixed. See the section on conformance for details.

XSL modifications to the CSS definition:

The last paragraph in the CSS description does not apply.

7.6.2. “background-color”

CSS2 Definition:

Value: <color> | transparent | inherit

Initial: transparent

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/colors.html#propdef-background-color>.

This property sets the background color of an element, either a <color> value or the keyword transparent, to make the underlying colors shine through.

transparent

The underlying colors will shine through.

<color>

Any valid color specification.

XSL modifications to the CSS definition:

XSL adds an "icc-color" function (see § 5.10.2 – Color Functions on page 60) as a valid value of this property.

7.6.3. “background-image”

CSS2 Definition:

Value: <uri-specification> | none | inherit

<i>Initial:</i>	none
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/colors.html#propdef-background-image>.

This property sets the background image of an element. When setting a "background-image", authors should also specify a background-color that will be used when the image is unavailable. When the image is available, it is rendered on top of the background color. (Thus, the color is visible in the transparent parts of the image).

Values for this property are either <uri-specification>, to specify the image, or "none", when no image is used.

none

No image is specified.

<uri-specification>

XSL modifications to the CSS definition:

The <uri> value has been changed to a <uri-specification>.

7.6.4. “background-repeat”

CSS2 Definition:

<i>Value:</i>	repeat repeat-x repeat-y no-repeat inherit
<i>Initial:</i>	repeat
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/colors.html#propdef-background-repeat>.

If a background image is specified, this property specifies whether the image is repeated (tiled), and how. All tiling covers the content and padding areas of a box. Values have the following meanings:

repeat

The image is repeated both horizontally and vertically.

repeat-x

The image is repeated horizontally only.

repeat-y

The image is repeated vertically only.

no-repeat

The image is not repeated: only one copy of the image is drawn.

XSL modifications to the CSS definition:

"Horizontal" and "vertical" are defined relative to the reference-orientation; "horizontal" is "left" to "right", and "vertical" is "top" to "bottom".

NOTE: Thus for a rotated area the tiling is also rotated. It is, however, independent of the writing-mode.

7.6.5. “background-position-horizontal”

A Property Derived from a CSS2 Property.

<i>Value:</i>	<percentage> <length> left center right inherit
<i>Initial:</i>	0%
<i>Applies to:</i>	all formatting objects to which background applies
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to the size of the padding-rectangle
<i>Media:</i>	visual

If a "background-image" has been specified, this property specifies its initial position horizontally.

<percentage>

Specifies that a point, at the given percentage across the image from left-to-right, shall be placed at a point at the given percentage across, from left-to-right, the area's padding-rectangle.

NOTE: For example with a value of 0%, the left-edge of the image is aligned with the left-edge of the area's padding-rectangle. A value of 100% places the right-edge of the image aligned with the right-edge of the padding-rectangle. With a value of 14%, a point 14% across the image is to be placed at a point 14% across the padding-rectangle.

<length>

Specifies that the left-edge of the image shall be placed at the specified length to the right of the left-edge of the padding-rectangle.

NOTE: For example with a value of 2cm, the left-edge of the image is placed 2cm to the right of the left-edge of the padding-rectangle.

left

Same as 0%.

center

Same as 50%.

right

Same as 100%.

7.6.6. “background-position-vertical”

A Property Derived from a CSS2 Property.

Value: <percentage> | <length> | top | center | bottom | inherit

Initial: 0%

Applies to: all formatting objects to which background applies

Inherited: no

Percentages: refer to the size of the padding-rectangle

Media: visual

If a "background-image" has been specified, this property specifies its initial position vertically.

<percentage>

Specifies that a point, at the given percentage down the image from top-to-bottom, shall be placed at a point at the given percentage down, from top-to-bottom, the area's padding-rectangle.

NOTE: For example with a value of 0%, the top-edge of the image is aligned with the top-edge of the area's padding-rectangle. A value of 100% places the bottom-edge of the image aligned with the bottom-edge of the padding-rectangle. With a value of 84%, a point 84% down the image is to be placed at a point 84% down the padding-rectangle.

<length>

Specifies that the top-edge of the image shall be placed at the specified length below the top-edge of the padding-rectangle.

NOTE: For example with a value of 2cm, the top-edge of the image is placed 2cm below the top-edge of the padding-rectangle.

top

Same as 0%.

center

Same as 50%.

bottom

Same as 100%.

7.6.7. “border-before-color”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<color> inherit
<i>Initial:</i>	the value of the 'color' property
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the color of the border on the before-edge of a block-area or inline-area.

See definition of property border-top-color (§ 7.6.19 – “border-top-color” on page 181).

7.6.8. “border-before-style”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<border-style> inherit
<i>Initial:</i>	none
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the border-style for the before-edge.

See definition of property border-top-style (§ 7.6.20 – “border-top-style” on page 182).

7.6.9. “border-before-width”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<border-width> <length-conditional> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the border-width for the before-edge.

See definition of property border-top-width (§ 7.6.21 – “border-top-width” on page 183).

XSL modifications to the CSS definition:

The following value type has been added for XSL:

<length-conditional>

A compound value specifying the width and any conditionality of the border for the before-edge.

The .length component is a <length>. The .conditionality component may be set to "discard" or "retain" to control if the border should be 0 or retained if its associated edge is a leading-edge in a reference-area for areas generated from this formatting object that have an *is-first* value of "false". See § 4.3 – [Spaces and Conditionality](#) on page 25 for further details. The initial value of the .conditionality component is "retain".

NOTE: If the border-style is "none" the computed value of the width is forced to "0pt".

7.6.10. “border-after-color”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<color> inherit
<i>Initial:</i>	the value of the 'color' property
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the color of the border on the after-edge of a block-area or inline-area.

See definition of property border-top-color (§ 7.6.19 – [“border-top-color”](#) on page 181).

7.6.11. “border-after-style”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<border-style> inherit
<i>Initial:</i>	none
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the border-style for the after-edge.

See definition of property border-top-style (§ 7.6.20 – [“border-top-style”](#) on page 182).

7.6.12. “border-after-width”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<border-width> <length-conditional> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the border-width for the after-edge.

See definition of property border-top-width (§ 7.6.21 – “border-top-width” on page 183).

XSL modifications to the CSS definition:

The following value type has been added for XSL:

<length-conditional>

A compound value specifying the width and any conditionality of the border for the after-edge.

The .length component is a <length>. The .conditionality component may be set to "discard" or "retain" to control if the border should be 0 or retained if its associated edge is a trailing-edge in a reference-area for areas generated from this formatting object that have an *is-last* value of "false". See § 4.3 – [Spaces and Conditionality](#) on page 25 for further details. The initial value of the .conditionality component is "retain".

NOTE: If the border-style is "none" the computed value of the width is forced to "0pt".

7.6.13. “border-start-color”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<color> inherit
<i>Initial:</i>	the value of the 'color' property
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the color of the border on the start-edge of a block-area or inline-area.

See definition of property border-top-color (§ 7.6.19 – “border-top-color” on page 181).

7.6.14. “border-start-style”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<border-style> inherit
<i>Initial:</i>	none
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the border-style for the start-edge.

See definition of property border-top-style (§ 7.6.20 – “border-top-style” on page 182).

7.6.15. “border-start-width”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<border-width> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the border-width for the start-edge.

NOTE: If the border-style is "none" the computed value of the width is forced to "0pt".

See definition of property border-top-width (§ 7.6.21 – “border-top-width” on page 183).

7.6.16. “border-end-color”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<color> inherit
<i>Initial:</i>	the value of the 'color' property
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the color of the border on the end-edge of a block-area or inline-area.

See definition of property border-top-color (§ 7.6.19 – “border-top-color” on page 181).

7.6.17. “border-end-style”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<border-style> inherit
<i>Initial:</i>	none
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the border-style for the end-edge.

See definition of property border-top-style (§ 7.6.20 – “border-top-style” on page 182).

7.6.18. “border-end-width”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<border-width> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the border-width for the end-edge.

NOTE: If the border-style is "none" the computed value of the width is forced to "0pt".

See definition of property border-top-width (§ 7.6.21 – “border-top-width” on page 183).

7.6.19. “border-top-color”

CSS2 Definition:

<i>Value:</i>	<color> inherit
<i>Initial:</i>	the value of the 'color' property
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-top-color>.

The 'border-color' property sets the color of the four borders. Values have the following meanings:

<color>

Any valid color specification.

If an element's border color is not specified with a "border" property, user agents must use the value of the element's "color" property as the computed value for the border color.

7.6.20. “border-top-style”

CSS2 Definition:

Value: <border-style> | inherit

Initial: none

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-top-style>.

The border style properties specify the line style of a box's border (solid, double, dashed, etc.).

The properties defined in this section refer to the <border-style> value type, which may take one of the following:

none

No border. This value forces the computed value of 'border-width' to be '0'.

hidden

Same as 'none', except in terms of border conflict resolution for table elements.

dotted

The border is a series of dots.

dashed

The border is a series of short line segments.

solid

The border is a single line segment.

double

The border is two solid lines. The sum of the two lines and the space between them equals the value of 'border-width'.

groove

The border looks as though it were carved into the canvas.

ridge

The opposite of 'groove': the border looks as though it were coming out of the canvas.

inset

The border makes the entire box look as though it were embedded in the canvas.

outset

The opposite of 'inset': the border makes the entire box look as though it were coming out of the canvas.

All borders are drawn on top of the box's background. The color of borders drawn for values of 'groove', 'ridge', 'inset', and 'outset' depends on the element's 'color' property.

Conforming HTML user agents may interpret 'dotted', 'dashed', 'double', 'groove', 'ridge', 'inset', and 'outset' to be 'solid'.

7.6.21. “border-top-width”

CSS2 Definition:

<i>Value:</i>	<border-width> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-top-width>.

The border width properties specify the width of the border area. The properties defined in this section refer to the <border-width> value type, which may take one of the following values:

thin

A thin border.

medium

A medium border.

thick

A thick border.

<length>

The border's thickness has an explicit value. Explicit border widths cannot be negative.

The interpretation of the first three values depends on the user agent. The following relationships must hold, however:

- 'thin' <='medium' <='thick'.
- Furthermore, these widths must be constant throughout a document.

7.6.22. “border-bottom-color”

CSS2 Definition:

<i>Value:</i>	<color> inherit
<i>Initial:</i>	the value of the 'color' property
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-bottom-color>.

Specifies the border color for the bottom-edge.

See definition of property border-top-color (§ 7.6.19 – “border-top-color” on page 181).

7.6.23. “border-bottom-style”

CSS2 Definition:

<i>Value:</i>	<border-style> inherit
<i>Initial:</i>	none
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-bottom-style>.

Specifies the border style for the bottom-edge.

See definition of property border-top-style (§ 7.6.20 – “border-top-style” on page 182).

7.6.24. “border-bottom-width”

CSS2 Definition:

<i>Value:</i>	<border-width> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no

Percentages: N/A
Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-bottom-width>.

Specifies the border width for the bottom-edge.

See definition of property border-top-width (§ 7.6.21 – “border-top-width” on page 183).

7.6.25. “border-left-color”

CSS2 Definition:

Value: <color> | inherit
Initial: the value of the 'color' property
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-left-color>.

Specifies the border color for the left-edge.

See definition of property border-top-color (§ 7.6.19 – “border-top-color” on page 181).

7.6.26. “border-left-style”

CSS2 Definition:

Value: <border-style> | inherit
Initial: none
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-left-style>.

Specifies the border style for the left-edge.

See definition of property border-top-style (§ 7.6.20 – “border-top-style” on page 182).

7.6.27. “border-left-width”

CSS2 Definition:

<i>Value:</i>	<border-width> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-left-width>.

Specifies the border width for the left-edge.

See definition of property border-top-width (§ 7.6.21 – “border-top-width” on page 183).

7.6.28. “border-right-color”

CSS2 Definition:

<i>Value:</i>	<color> inherit
<i>Initial:</i>	the value of the 'color' property
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-right-color>.

Specifies the border color for the right-edge.

See definition of property border-top-color (§ 7.6.19 – “border-top-color” on page 181).

7.6.29. “border-right-style”

CSS2 Definition:

<i>Value:</i>	<border-style> inherit
<i>Initial:</i>	none
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-right-style>.

Specifies the border style for the right-edge.

See definition of property border-top-style (§ 7.6.20 – “border-top-style” on page 182).

7.6.30. “border-right-width”

CSS2 Definition:

<i>Value:</i>	<border-width> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-right-width>.

Specifies the border width for the right-edge.

See definition of property border-top-width (§ 7.6.21 – “border-top-width” on page 183).

7.6.31. “padding-before”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<padding-width> <length-conditional> inherit
<i>Initial:</i>	0pt
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

Specifies the width of the padding on the before-edge of a block-area or inline-area.

See definition of property padding-top (§ 7.6.35 – “padding-top” on page 189).

XSL modifications to the CSS definition:

The following value type has been added for XSL:

<length-conditional>

A compound value specifying the width and any conditionality of the padding for the before-edge.

The .length component is a <length>. The .conditionality component may be set to "discard" or "retain" to control if the padding should be 0 or retained if its associated edge is a leading-edge in a reference-area for areas generated from this formatting object that have an *is-first* value of "false". See § 4.3 – [Spaces and Conditionality](#) on page 25 for further details. The initial value of

the .conditionality component is "retain".

7.6.32. “padding-after”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<padding-width> <length-conditional> inherit
<i>Initial:</i>	Opt
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

Specifies the width of the padding on the after-edge of a block-area or inline-area.

See definition of property padding-top (§ 7.6.35 – “padding-top” on page 189).

XSL modifications to the CSS definition:

The following value type has been added for XSL:

<length-conditional>

A compound value specifying the width and any conditionality of the padding for the after-edge.

The .length component is a <length>. The .conditionality component may be set to "discard" or "retain" to control if the padding should be 0 or retained if its associated edge is a trailing-edge in a reference-area for areas generated from this formatting object that have an *is-last* value of "false". See § 4.3 – [Spaces and Conditionality](#) on page 25 for further details. The initial value of the .conditionality component is "retain".

7.6.33. “padding-start”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<padding-width> inherit
<i>Initial:</i>	Opt
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

Specifies the width of the padding on the start-edge of a block-area or inline-area.

See definition of property padding-top (§ 7.6.35 – “padding-top” on page 189).

7.6.34. “padding-end”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	<padding-width> inherit
<i>Initial:</i>	0pt
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

Specifies the width of the padding on the end-edge of a block-area or inline-area.

See definition of property padding-top (§ 7.6.35 – “padding-top” on page 189).

7.6.35. “padding-top”

CSS2 Definition:

<i>Value:</i>	<padding-width> inherit
<i>Initial:</i>	0pt
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-padding-top>.

<length>

Specifies the width of the padding on the top-edge of a block-area or inline-area.

7.6.36. “padding-bottom”

CSS2 Definition:

<i>Value:</i>	<padding-width> inherit
<i>Initial:</i>	0pt
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-padding-bottom>.

Specifies the width of the padding on the bottom-edge of a block-area or inline-area.

See definition of property padding-top (§ 7.6.35 – “padding-top” on page 189).

7.6.37. “padding-left”

CSS2 Definition:

<i>Value:</i>	<padding-width> inherit
<i>Initial:</i>	0pt
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-padding-left>.

Specifies the width of the padding on the left-edge of a block-area or inline-area.

See definition of property padding-top (§ 7.6.35 – “padding-top” on page 189).

7.6.38. “padding-right”

CSS2 Definition:

<i>Value:</i>	<padding-width> inherit
<i>Initial:</i>	0pt
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-padding-right>.

Specifies the width of the padding on the right-edge of a block-area or inline-area.

See definition of property padding-top (§ 7.6.35 – “padding-top” on page 189).

7.7. Common Font Properties

The following common-font-properties all are taken from CSS2. The reference to CSS2 is: <http://www.w3.org/TR/REC-CSS2/fonts.html>

NOTE: Although these properties reference the individual properties in the CSS specification, it is recommended that you read the entire font section of the CSS2 specification.

7.7.1. Fonts and Font Data

XSL uses an abstract model of a font. This model is described in this section and is based on current

font technology as exemplified by the OpenType specification [[OpenType](#)].

A font consists of a collection of glyphs together with the information, the font tables, necessary to use those glyphs to present characters on some medium. A glyph is a recognizable abstract graphic symbol which is independent of any specific design. The combination of the collection of glyphs and the font tables is called the font data.

The font tables include the information necessary to map characters to glyphs, to determine the size of glyph areas and to position the glyph area. Each font table consists of one or more font characteristics, such as the font-weight and font-style.

The geometric font characteristics are expressed in a coordinate system based on the EM box. (The EM is a relative measure of the height of the glyphs in the font; see § 5.9.7.2 – [Relative Lengths](#) on page 56.) This box that is 1 EM high and 1 EM wide is called the design space. Points in this design space are expressed in geometric coordinates in terms of fractional units of the EM.

The coordinate space of the EM box is called the design space coordinate system. For scalable fonts, the curves and lines that are used to draw a glyph are represented using this coordinate system.

NOTE: Most often, the (0,0) point in this coordinate system is positioned on the left edge of the EM box, but not at the bottom left corner. The Y coordinate of the bottom of a Roman capital letter is usually zero. In addition, the descenders on lower case Roman letters have negative coordinate values.

XSL assumes that the font tables will provide at least three font characteristics: an ascent, a descent and a set of baseline-tables. The coordinate values for these are given in the design space coordinate system. The ascent is given by the vertical coordinate of the top of the EM box; the descent is given by the vertical coordinate of the bottom of the EM box. The baseline-table is explained below.

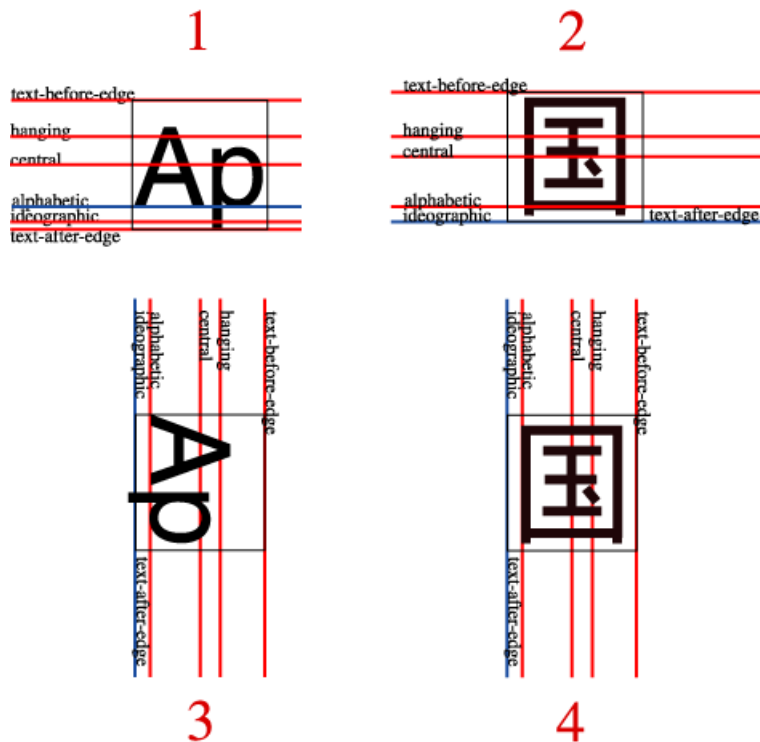
The glyphs of a given script are positioned so that a particular point on each glyph, the *alignment-point*, is aligned with the alignment-points of the other glyphs in that script. The glyphs of different scripts are typically aligned at different points on the glyph. For example, Western glyphs are aligned on the bottoms of the capital letters, certain Indic glyphs (including glyphs from the Devanagari, Gurmukhi and Bengali scripts) are aligned at the top of a horizontal stroke near the top of the glyphs and Far-Eastern glyphs are aligned either at the bottom or center of the EM box of the glyph. Within a script and within a line of text having a single font-size, the sequence of alignment-points defines, in the inline-progression-direction, a geometric line called a *baseline*. Western and most other alphabetic and syllabic glyphs are aligned to an "alphabetic" baseline, the above Indic glyphs are aligned to a "hanging" baseline and the Far-Eastern glyphs are aligned to an "ideographic" baseline.



This figure shows the vertical position of the alignment-point for alphabetic and many syllabic scripts, illustrated by a Roman "A"; for certain Indic scripts, illustrated by a Gurmukhi syllable "ji"; and for ideographic scripts, illustrated by the ideographic glyph meaning "country". The thin black rectangle around the ideographic glyph illustrates the EM box for that glyph and shows the typical positioning of the "black marks" of the glyph within the EM box.

A *baseline-table* specifies the position of one or more baselines in the design space coordinate system. The function of the baseline table is to facilitate the alignment of different scripts with respect to each other when they are mixed on the same text line. Because the desired relative alignments may depend on

which script is dominant in a line (or block), there may be a different baseline table for each script. In addition, different alignment positions are needed for horizontal and vertical writing modes. Therefore, the font may have a set of baseline tables: typically, one or more for horizontal writing-modes and zero or more for vertical writing-modes.

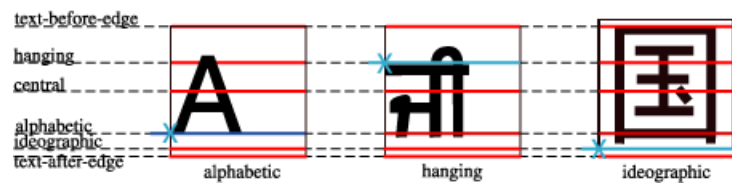


Examples of horizontal and vertical baseline positions. The thin lined box in each example is the "EM box". For the Latin glyphs, only the EM box of the first glyph is shown. Example 1 shows typical Latin text written horizontally. This text is positioned relative to the alphabetic baseline, shown in blue. Example 2 shows a typical ideographic glyph positioned on the horizontal ideographic baseline. Note that the EM Box is positioned differently for these two cases. Examples 3 and 4 show the same set of baselines used in vertical writing. The Latin text, example 3, is shown with a glyph-orientation of 90 degrees which is typical for proportionally space Latin glyphs in vertical writing. Even though the ideographic glyph in Example 4 is positioned on the vertical ideographic baseline, because it is centered in the EM box, all glyphs with the same EM Box are centered, vertically, with respect to one another. Additional examples showing the positioning of mixed scripts are given in the introductions to [§ 7.11 – Area Alignment Properties](#) on page 212 and [§ 7.25 – Writing-mode-related Properties](#) on page 307.

The font tables for a font include font characteristics for the individual glyphs in the font. XSL assumes that the font tables include, for each glyph in the font, one width value, one alignment-baseline and one alignment-point for the horizontal writing-modes. If vertical writing-modes are supported, then each glyph must have another width value, alignment-baseline and alignment-point for the vertical writing-modes. (Even though it is specified as a width, for vertical writing-modes the width is used in the vertical direction.)

The script to which a glyph belongs determines an alignment-baseline to which the glyph is to be aligned. The position of this baseline in the design space coordinate system determines the default block-progression-direction position of the alignment-point. The inline-progression-direction position of the alignment-point is on the start-edge of the glyph. (These positions are adjusted according to the

specifications in § 7.11.1 – “alignment-adjust” on page 221 when an instance of a glyph is used in an inline or block formatting object. The "space-start" and/or the "space-end" properties of the fo:character that maps to the glyph may be adjusted to effect "kerning" with respect to adjacent glyphs.)



This figure shows glyphs from three different scripts, each with its EM box and within the EM box, the baseline table applicable to that glyph. The alignment-point of each glyph is shown by an "X" on the start edge of the EM box and by making alignment-baseline blue. The baseline-table of the parent formatting object of the characters that mapped to these glyphs is shown as a set of dashed lines.

In addition to the font characteristics required above, a font may also supply substitution and positioning tables that can be used by a formatter to re-order, combine, and position a sequence of glyphs to make one or more composite glyphs. The combination may be as simple as a ligature, or as complex as an Indic syllable which combines, usually with some re-ordering, multiple consonants and vowel glyphs. See § 4.7.2 – Line-building on page 33.

NOTE: If the font tables do not define values for required font characteristics, heuristics may be used to approximate these values.

7.7.2. “font-family”

CSS2 Definition:

<i>Value:</i>	<code>[[<family-name> <generic-family>],]* [<family-name> <generic-family>] inherit</code>
<i>Initial:</i>	depends on user agent
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/fonts.html#propdef-font-family>.

This property specifies a prioritized list of font family names and/or generic family names. To deal with the problem that a single font may not contain glyphs to display all the characters in a document, or that not all fonts are available on all systems, this property allows authors to specify a list of fonts, all of the same style and size, that are tried in sequence to see if they contain a glyph for a certain character. This list is called a font set.

The generic font family will be used if one or more of the other fonts in a font set is unavailable. Although many fonts provide the "missing character" glyph, typically an open box, as its name implies this should not be considered a match except for the last font in a font set.

There are two types of font family names:

<family-name>

The name of a font-family of choice. In the previous example [in the CSS2 Recommendation], "Baskerville", "Heisi Mincho W3", and "Symbol" are font families. Font family names containing whitespace should be quoted. If quoting is omitted, any whitespace characters before and after the font name are ignored and any sequence of whitespace characters inside the font name is converted to a single space.

<generic-family>

The following generic families are defined: "serif", "sans-serif", "cursive", "fantasy", and "monospace". Please see the section on generic font families for descriptions of these families. Generic font family names are keywords, and therefore must not be quoted.

XSL modifications to the CSS definition:

<string>

The names are syntactically expressed as strings.

NOTE: See the expression language for a two-argument "system-font" function that returns a characteristic of a system-font. This may be used, instead of the "font" shorthand, to specify the name of a system-font.

7.7.3. “font-selection-strategy”

XSL Definition:

<i>Value:</i>	auto character-by-character inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

There is no XSL mechanism to specify a particular font; instead, a *selected font* is chosen from the fonts available to the User Agent based on a set of selection criteria. The *selection criteria* are the following font properties: "font-family", "font-style", "font-variant", "font-weight", "font-stretch", and "font-size", plus, for some formatting objects, one or more characters. These characters are called the *contextual characters*. The contextual characters can be as few as a single character and as many as the entire character complement of the result tree being processed.

Except for the fo:character formatting object, for all other formatting objects where "font-family" applies, the selection criteria consist of the above font properties only. For the fo:character formatting object, the selection criteria are these properties plus either the value of the "character" property of the fo:character alone or that character together with other contextual characters.

The strategy to be followed for selecting a font based on these criteria is specified by the "font-selection-strategy" property.

The "font-family" property is a prioritized list of font family names, which are tried in sequence to find an available font that matches the selection criteria. The font property selection criteria are matched if the corresponding font characteristics match the properties as specified in the property descriptions.

If no matching font is found, a fallback selection is determined in a system-dependent manner.

NOTE: This fallback may be to seek a match using a User Agent default "font-family", or it may be a more elaborate fallback strategy where, for example, "Helvetica" would be used as a fallback for "Univers".

If no match has been found for a particular character, there is no selected font and the User Agent should indicate that a character is not being displayed (for example, using the 'missing character' glyph).

Values of the "font-selection-strategy" property have the following meanings:

auto

The selection criterion given by the contextual characters is used in an implementation defined manner.

NOTE: An implementation may, for example, use an algorithm where all characters in the result tree having the same set of font selection property values influence the selection, or it may only use the character property of a single fo:character formatting object for which a font is to be selected. Consider, for example, a case where the available fonts include a font that covers all of Latin, Greek and Cyrillic as well as three better quality fonts that cover those three separately, but match each other badly stylistically. An implementation that takes a larger view for its set of contextual characters may consider the glyph complement to allow the selection of the better font if it covers the glyph complement, but to use the broader font to get a consistent style if the glyph complement is larger than any one of the other fonts can cover.

character-by-character

The set of contextual characters consists of the single character that is the value of the "character" property of the fo:character for which a font is to be selected.

NOTE: This selection strategy is the same as the strategy used to select fonts in CSS.

Describes the criteria for selecting fonts and the different strategies for using these criteria to determine a *selected font*.

7.7.4. “font-size”

CSS2 Definition:

<i>Value:</i>	<absolute-size> <relative-size> <length> <percentage> inherit
<i>Initial:</i>	medium
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes, the computed value is inherited
<i>Percentages:</i>	refer to parent element's font size
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/fonts.html#propdef-font-size>.

This property describes the size of the font when set solid. Values have the following meanings:

<absolute-size>

An <absolute-size> keyword refers to an entry in a table of font sizes computed and kept by the user agent. Possible values are:

[xx-small | x-small | small | medium | large | x-large | xx-large]

On a computer screen a scaling factor of 1.2 is suggested between adjacent indexes; if the "medium" font is 12pt, the "large" font could be 14.4pt. Different media may need different scaling factors. Also, the user agent should take the quality and availability of fonts into account when computing the table. The table may be different from one font family to another. Note. In CSS1, the suggested scaling factor between adjacent indexes was 1.5 which user experience proved to be too large.

<relative-size>

A <relative-size> keyword is interpreted relative to the table of font sizes and the font size of the parent element. Possible values are:

[larger | smaller]

For example, if the parent element has a font size of "medium", a value of "larger" will make the font size of the current element be "large". If the parent element's size is not close to a table entry, the user agent is free to interpolate between table entries or round off to the closest one. The user agent may have to extrapolate table values if the numerical value goes beyond the keywords.

<length>

A length value specifies an absolute font size (that is independent of the user agent's font table). Negative lengths are illegal.

<percentage>

A percentage value specifies an absolute font size relative to the parent element's font size. Use of percentage values, or values in "em's", leads to more robust and cascadable stylesheets.

The actual value of this property may differ from the computed value due a numerical value on 'font-size-adjust' and the unavailability of certain font sizes.

Child elements inherit the computed 'font-size' value (otherwise, the effect of 'font-size-adjust' would compound).

XSL modifications to the CSS definition:

XSL incorporates the following text from CSS2 15.5 (<http://www.w3.org/TR/REC-CSS2/fonts.html#algorithm>) as part of the property definition.

'font-size' must be matched within a UA-dependent margin of tolerance. (Typically, sizes for scalable fonts are rounded the nearest whole pixel, while the tolerance for bitmapped fonts could be as large as 20%.) Further computations, e.g., by 'em' values in other properties, are based on the computed 'font-size' value.

7.7.5. “font-stretch”

CSS2 Definition:

<i>Value:</i>	normal wider narrower ultra-condensed extra-condensed condensed semi-condensed semi-expanded expanded extra-expanded ultra-expanded inherit
<i>Initial:</i>	normal
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/fonts.html#font-styling>.

The 'font-stretch' property selects a normal, condensed, or extended face from a font family.

ultra-condensed

extra-condensed

condensed

semi-condensed

normal

semi-expanded

expanded

extra-expanded

ultra-expanded

Absolute keyword values have the following ordering, from narrowest to widest :

1. ultra-condensed
2. extra-condensed
3. condensed
4. semi-condensed
5. normal
6. semi-expanded
7. expanded
8. extra-expanded
9. ultra-expanded

wider

The relative keyword "wider" sets the value to the next expanded value above the inherited value (while not increasing it above "ultra-expanded").

narrower

The relative keyword "narrower" sets the value to the next condensed value below the inherited value (while not decreasing it below "ultra-condensed").

7.7.6. “font-size-adjust”

CSS2 Definition:

<i>Value:</i>	<number> none inherit
<i>Initial:</i>	none
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/fonts.html#font-size-props>.

In bicameral scripts, the subjective apparent size and legibility of a font are less dependent on their 'font-size' value than on the value of their 'x-height', or, more usefully, on the ratio of these two values, called the aspect value (font size divided by x-height). The higher the aspect value, the more likely it is that a font at smaller sizes will be legible. Inversely, faces with a lower aspect value will become illegible more rapidly below a given threshold size than faces with a higher aspect value. Straightforward font substitution that relies on font size alone may lead to illegible characters.

For example, the popular font Verdana has an aspect value of 0.58; when Verdana's font size 100 units, its x-height is 58 units. For comparison, Times New Roman has an aspect value of 0.46. Verdana will therefore tend to remain legible at smaller sizes than Times New Roman. Conversely, Verdana will often look 'too big' if substituted for Times New Roman at a chosen size.

This property allows authors to specify an aspect value for an element that will preserve the x-height of the first choice font in the substitute font. Values have the following meanings:

none

Do not preserve the font's x-height.

<number>

Specifies the aspect value. The number refers to the aspect value of the first choice font. The scaling factor for available fonts is computed according to the following formula:

$$y(a/a') = c$$

where:

y="font-size" of first-choice font

a' = aspect value of available font

c="font-size" to apply to available font

This property allows authors to specify an aspect value for an element that will preserve the x-height of the first choice font in the substitute font.

Font size adjustments take place when computing the actual value of "font-size". Since inheritance is based on the computed value, child elements will inherit unadjusted values.

7.7.7. "font-style"

CSS2 Definition:

Value: normal | italic | oblique | backslant | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/fonts.html#font-styling>.

The "font-style" property requests normal (sometimes referred to as "roman" or "upright"), italic, and oblique faces within a font family. Values have the following meanings:

normal

Specifies a font that is classified as "normal" in the UA's font database.

oblique

Specifies a font that is classified as "oblique" in the UA's font database. Fonts with Oblique, Slanted, or Incline in their names will typically be labeled "oblique" in the font database. A font that is labeled "oblique" in the UA's font database may actually have been generated by electronically slanting a normal font.

italic

Specifies a font that is classified as "italic" in the UA's font database, or, if that is not available, one labeled 'oblique'. Fonts with Italic, Cursive, or Kursiv in their names will typically be labeled "italic".

XSL modifications to the CSS definition:

The following value type has been added for XSL:

backslant

Specifies a font that is classified as "backslant" in the UA's font database.

XSL incorporates the following text from CSS2 15.5

(<http://www.w3.org/TR/REC-CSS2/fonts.html#algorithm>) as part of the property definition, except that for XSL the information is obtained from the font tables of the available fonts.

'italic' will be satisfied if there is either a face in the UA's font database labeled with the CSS keyword 'italic' (preferred) or 'oblique'. Otherwise the values must be matched exactly or font-style will fail.

7.7.8. “font-variant”

CSS2 Definition:

<i>Value:</i>	normal small-caps inherit
<i>Initial:</i>	normal
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/fonts.html#font-styling>.

In a small-caps font, the glyphs for lowercase letters look similar to the uppercase ones, but in a smaller size and with slightly different proportions. The "font-variant" property requests such a font for bicameral (having two cases, as with Roman script). This property has no visible effect for scripts that are unicameral (having only one case, as with most of the world's writing systems). Values have the following meanings:

normal

Specifies a font that is not labeled as a small-caps font.

small-caps

Specifies a font that is labeled as a small-caps font. If a genuine small-caps font is not available, user agents should simulate a small-caps font, for example by taking a normal font and replacing the lowercase letters by scaled uppercase characters. As a last resort, unscaled uppercase letter glyphs in a normal font may replace glyphs in a small-caps font so that the text appears in all uppercase letters.

Insofar as this property causes text to be transformed to uppercase, the same considerations as for "text-transform" apply.

XSL modifications to the CSS definition:

XSL incorporates the following text from CSS2 15.5 (<http://www.w3.org/TR/REC-CSS2/fonts.html#algorithm>) as part of the property definition.

'normal' matches a font not labeled as 'small-caps'; 'small-caps' matches (1) a font labeled as 'small-caps', (2) a font in which the small caps are synthesized, or (3) a font where all lowercase letters are replaced by uppercase letters. A small-caps font may be synthesized by electronically scaling uppercase letters from a normal font.

7.7.9. “font-weight”

CSS2 Definition:

<i>Value:</i>	normal bold bolder lighter 100 200 300 400 500 600 700 800 900 inherit
<i>Initial:</i>	normal
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/fonts.html#font-styling>.

The “font-weight” property specifies the weight of the font.

normal

Same as “400”.

bold

Same as “700”.

bolder

Specifies the next weight that is assigned to a font that is darker than the inherited one. If there is no such weight, it simply results in the next darker numerical value (and the font remains unchanged), unless the inherited value was “900”, in which case the resulting weight is also “900”.

lighter

Specifies the next weight that is assigned to a font that is lighter than the inherited one. If there is no such weight, it simply results in the next lighter numerical value (and the font remains unchanged), unless the inherited value was “100”, in which case the resulting weight is also “100”.

<integer>

These values form an ordered sequence, where each number indicates a weight that is at least as dark as its predecessor.

Child elements inherit the computed value of the weight.

XSL modifications to the CSS definition:

XSL incorporates the following text from CSS2 15.5.1 (<http://www.w3.org/TR/REC-CSS2/fonts.html#q46>) as part of the property definition.

The association of other weights within a family to the numerical weight values is intended only to preserve the ordering of weights within that family. User agents must map names to values in a way that preserves visual order; a face mapped to a value must not be lighter than faces mapped to lower values. There is no guarantee on how a user agent will map fonts within a family to weight values. However,

the following heuristics tell how the assignment is done in typical cases: If the font family already uses a numerical scale with nine values (as e.g., OpenType does), the font weights should be mapped directly.

If there is both a face labeled Medium and one labeled Book, Regular, Roman or Normal, then the Medium is normally assigned to the '500'.

The font labeled "Bold" will often correspond to the weight value '700'.

If there are fewer than 9 weights in the family, the default algorithm for filling the "holes" is as follows. If '500' is unassigned, it will be assigned the same font as '400'. If any of the values '600', '700', '800', or '900' remains unassigned, they are assigned to the same face as the next darker assigned keyword, if any, or the next lighter one otherwise. If any of '300', '200', or '100' remains unassigned, it is assigned to the next lighter assigned keyword, if any, or the next darker otherwise.

There is no guarantee that there will be a darker face for each of the 'font-weight' values; for example, some fonts may have only a normal and a bold face, others may have eight different face weights.

7.8. Common Hyphenation Properties

7.8.1. “country”

XSL Definition:

<i>Value:</i>	none <country> inherit
<i>Initial:</i>	none
<i>Applies to:</i>	fo:block, fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

none

Indicates the country is unknown or is not significant to the proper formatting of this object.

<country>

A country-specifier in conformance with [\[RFC1766\]](#).

Specifies the country to be used by the formatter in language-/locale-coupled services, such as line-justification strategy, line-breaking, and hyphenation.

NOTE: This may affect line composition in a system-dependent way.

The country may be the country component of any RFC 1766 code; (these are derived from ISO 3166).

7.8.2. “language”

XSL Definition:

<i>Value:</i>	none <language> inherit
<i>Initial:</i>	none

<i>Applies to:</i>	fo:block, fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

none

Indicates the language is unknown or is not significant to the proper formatting of this object.

<language>

A language-specifier in conformance with [\[RFC1766\]](#).

Specifies the language to be used by the formatter in language-/locale-coupled services, such as line-justification strategy, line-breaking, and hyphenation.

NOTE: This may affect line composition in a system-dependent way.

The language may be the language component of any RFC 1766 code (these are derived from the ISO 639 language codes).

7.8.3. “script”

XSL Definition:

<i>Value:</i>	none auto <script> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:block, fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

Indicates that the script is determined using codepoint ranges in the text of the document.

NOTE: This provides the automatic differentiation between Kanji, Katakana, Hiragana, and Romanji used in JIS-4051 and similar services in some other countries/languages.

none

Indicates the script is unknown or is not significant to the proper formatting of this object.

<script>

A script specifier in conformance with [\[ISO15924\]](#).

Specifies the script to be used by the formatter in language-/locale-coupled services, such as

line-justification strategy, line-breaking, and hyphenation.

NOTE: This may affect line composition in a system-dependent way.

The script may be any ISO 15924 script code.

7.8.4. “hyphenate”

XSL Definition:

<i>Value:</i>	false true inherit
<i>Initial:</i>	false
<i>Applies to:</i>	fo:block, fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

false

Hyphenation may not be used in the line-breaking algorithm for the text contained in this object.

true

Hyphenation may be used in the line-breaking algorithm for the text contained in this object.

Specifies whether hyphenation is allowed during line-breaking when the formatter is formatting this formatting object.

7.8.5. “hyphenation-character”

XSL Definition:

<i>Value:</i>	<character> inherit
<i>Initial:</i>	The Unicode hyphen character u+2010
<i>Applies to:</i>	fo:block, fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<character>

Specifies the Unicode character to be presented when a hyphenation break occurs. The styling properties of this character are those inherited from its containing flow object.

7.8.6. “hyphenation-push-character-count”

XSL Definition:

<i>Value:</i>	<number> inherit
<i>Initial:</i>	2
<i>Applies to:</i>	fo:block, fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<integer>

If a negative or non-integer value is specified, it will be rounded to the nearest integer greater than zero.

The hyphenation-push-character-count is a positive integer specifying the minimum number of characters in a hyphenated word after the hyphenation character. This is the minimum number of characters in the word pushed to the next line after the line ending with the hyphenation character.

7.8.7. “hyphenation-remain-character-count”

XSL Definition:

<i>Value:</i>	<number> inherit
<i>Initial:</i>	2
<i>Applies to:</i>	fo:block, fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<integer>

If a negative or non-integer value is specified, it will be rounded to the nearest integer greater than zero.

The hyphenation-remain-character-count is a positive integer specifying the minimum number of characters in a hyphenated word before the hyphenation character. This is the minimum number of characters in the word left on the line ending with the hyphenation character.

7.9. Common Margin Properties-Block

7.9.1. “margin-top”

CSS2 Definition:

<i>Value:</i>	<margin-width> inherit
<i>Initial:</i>	0
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-margin-top>.

Margin-width may be one of the following:

auto

See the CSS2 section on computing widths and margins for behavior.

<length>

Specifies a fixed width.

<percentage>

The percentage is calculated with respect to the width of the generated box's containing block. This is true for 'margin-top' and 'margin-bottom', except in the page context, where percentages refer to page box height.

Negative values for margin properties are allowed, but there may be implementation-specific limits.

Sets the top margin of a box.

XSL modifications to the CSS definition:

- Margin-top is provided for compatibility with CSS.
- Details on the mapping of CSS "margin" properties for XSL are given in [Property Refinement / Resolution](#) — § 5 on page 39.

7.9.2. “margin-bottom”

CSS2 Definition:

<i>Value:</i>	<margin-width> inherit
<i>Initial:</i>	0
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-margin-bottom>.

Margin-width may be one of the following:

auto

See the CSS2 section on computing widths and margins for behavior.

<length>

Specifies a fixed width.

<percentage>

The percentage is calculated with respect to the width of the generated box's containing block. This is true for 'margin-top' and 'margin-bottom', except in the page context, where percentages refer to page box height.

Negative values for margin properties are allowed, but there may be implementation-specific limits.

Sets the bottom margin of a box.

XSL modifications to the CSS definition:

- Margin-bottom is provided for compatibility with CSS.
- Details on the mapping of CSS "margin" properties for XSL are given in [Property Refinement / Resolution](#) — § 5 on page 39.

7.9.3. “margin-left”

CSS2 Definition:

Value: <margin-width> | inherit

Initial: 0pt

Applies to: all elements

Inherited: no

Percentages: refer to width of containing block

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-margin-left>.

Margin-width may be one of the following:

auto

See the CSS2 section on computing widths and margins for behavior.

<length>

Specifies a fixed width.

<percentage>

The percentage is calculated with respect to the width of the generated box's containing block.

Negative values for margin properties are allowed, but there may be implementation-specific limits.

Sets the left margin of a box.

XSL modifications to the CSS definition:

- Margin-left is provided for compatibility with CSS.
- Details on the mapping of CSS "margin" properties for XSL are given in [Property Refinement / Resolution](#) — § 5 on page 39.

7.9.4. “margin-right”

CSS2 Definition:

Value: <margin-width> | inherit
Initial: 0pt
Applies to: all elements
Inherited: no
Percentages: refer to width of containing block
Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-margin-right>.

Margin-width may be one of the following:

auto

See the CSS2 section on computing widths and margins for behavior.

<length>

Specifies a fixed width.

<percentage>

The percentage is calculated with respect to the width of the generated box's containing block.

Negative values for margin properties are allowed, but there may be implementation-specific limits.

Sets the right margin of a box.

XSL modifications to the CSS definition:

- Margin-right is provided for compatibility with CSS.
- Details on the mapping of CSS "margin" properties for XSL are given in [Property Refinement / Resolution](#) — § 5 on page 39.

7.9.5. “space-before”

XSL Definition:

<i>Value:</i>	<space> inherit
<i>Initial:</i>	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0
<i>Applies to:</i>	all block-level formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A (Differs from margin-top in CSS)
<i>Media:</i>	visual

Values have the following meanings:

<space>

Specifies the minimum, optimum, and maximum values for the space before any areas generated by this formatting object and the conditionality and precedence of this space.

Specifies the value of the space-specifier for the space before the areas generated by this formatting object. A definition of space-specifiers, and the interaction between space-specifiers occurring in sequence are given in § 4.3 – Spaces and Conditionality on page 25.

NOTE: A common example of such a sequence is the "space-after" on one area and the "space-before" of its next sibling.

7.9.6. “space-after”

XSL Definition:

<i>Value:</i>	<space> inherit
<i>Initial:</i>	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0
<i>Applies to:</i>	all block-level formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A (Differs from margin-bottom in CSS)
<i>Media:</i>	visual

Values have the following meanings:

<space>

Specifies the minimum, optimum, and maximum values for the space after any areas generated by this formatting object and the conditionality and precedence of this space.

Specifies the value of the space-specifier for the space after the areas generated by this formatting object. A definition of space-specifiers, and the interaction between space-specifiers occurring in sequence are given in § 4.3 – Spaces and Conditionality on page 25.

NOTE: A common example of such a sequence is the "space-after" on one area and the "space-before" of its next

sibling.

7.9.7. “start-indent”

XSL Definition:

<i>Value:</i>	<length> inherit
<i>Initial:</i>	Opt
<i>Applies to:</i>	all block-level formatting objects
<i>Inherited:</i>	yes
<i>Percentages:</i>	refer to width of containing reference-area
<i>Media:</i>	visual

Values have the following meanings:

<length>

For each block-area generated by this formatting object, specifies the distance from the start-edge of the content-rectangle of the containing reference-area to the start-edge of the content-rectangle of that block-area.

This property may have a negative value, which indicates an outdent.

7.9.8. “end-indent”

XSL Definition:

<i>Value:</i>	<length> inherit
<i>Initial:</i>	Opt
<i>Applies to:</i>	all block-level formatting objects
<i>Inherited:</i>	yes
<i>Percentages:</i>	refer to width of containing reference-area
<i>Media:</i>	visual

Values have the following meanings:

<length>

For each block-area generated by this formatting object, specifies the distance from the end-edge of the content-rectangle of that block-area to the end-edge of the content-rectangle of the containing reference-area.

This property may have a negative value, which indicates an outdent.

7.10. Common Margin Properties-Inline

This group also includes all the properties in the common-margin-properties-block group except space-before, space-after, start-indent, and end-indent.

7.10.1. “space-end”

XSL Definition:

<i>Value:</i>	<space> inherit
<i>Initial:</i>	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0
<i>Applies to:</i>	all inline-level formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to the width of the containing area
<i>Media:</i>	visual

Values have the following meanings:

<space>

Specifies the minimum, optimum, and maximum values for the space after any areas generated by this formatting object and the conditionality and precedence of this space.

Specifies the value of the space-specifier for the space after the areas generated by this formatting object. A definition of space-specifiers, and the interaction between space-specifiers occurring in sequence are given in § 4.3 – Spaces and Conditionality on page 25.

NOTE: A common example of such a sequence is the "space-end" on one area and the "space-start" of its next sibling.

7.10.2. “space-start”

XSL Definition:

<i>Value:</i>	<space> inherit
<i>Initial:</i>	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0
<i>Applies to:</i>	all inline-level formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to the width of the containing area
<i>Media:</i>	visual

Values have the following meanings:

<space>

Specifies the minimum, optimum, and maximum values for the space before any areas generated by this formatting object and the conditionality and precedence of this space.

Specifies the value of the space-specifier for the space before the areas generated by this formatting object. A definition of space-specifiers, and the interaction between space-specifiers occurring in sequence are given in § 4.3 – Spaces and Conditionality on page 25.

NOTE: A common example of such a sequence is the "space-end" on one area and the "space-start" of its next

sibling.

7.11. Area Alignment Properties

The area alignment properties control the alignment of child areas with respect to their parent areas. The parent area is given a frame of reference through its scaled-baseline-table.

The positions of these baselines are illustrated in the following figure:



This figure shows samples of Gurmukhi (a hanging Indic script), Latin and ideographic scripts together with most of the baselines defined below. The thin line around the ideographic glyphs symbolizes the EM box in which these glyphs are centered. In this figure, the position of the "text-before-edge" and "text-after-edge" baselines is computed assuming that the "alphabetic" baseline is the dominant-baseline. The "central" baseline has been omitted from the figure, but it lies halfway between the "text-before-edge" and "text-after-edge" baselines, just about where the "math" baseline is shown.

The baseline-identifiers below are used in this specification. Some of these are determined by baseline-tables contained in a font as described in § 7.7.1 – [Fonts and Font Data](#) on page 190. Others are computed from other font characteristics as described below.

alphabetic

This identifies the baseline used by most alphabetic and syllabic scripts. These include, but are not limited to, many Western, Southern Indic, Southeast Asian (non-ideographic) scripts.

ideographic

This identifies the baseline used by ideographic scripts. For historical reasons, this baseline is at the bottom of the ideographic EM box and not in the center of the ideographic EM box. See the "central" baseline. The ideographic scripts include Chinese, Japanese, Korean, and Vietnamese Chu Nom.

hanging

This identifies the baseline used by certain Indic scripts. These scripts include Devanagari, Gurmukhi and Bengali.

mathematical

This identifies the baseline used by mathematical symbols.

central

This identifies a computed baseline that is at the center of the EM box. This baseline lies halfway between the text-before-edge and text-after-edge baselines.

NOTE: For ideographic fonts, this baseline is often used to align the glyphs; it is an alternative to the ideographic baseline.

middle

This identifies a baseline that is offset from the alphabetic baseline in the *shift-direction* by 1/2 the value of the x-height font characteristic. The position of this baseline may be obtained from the font data or, for fonts that have a font characteristic for "x-height", it may be computed using 1/2 the "x-height". Lacking either of these pieces of information, the position of this baseline may be approximated by the "central" baseline.

text-before-edge

This identifies the before-edge of the EM box. The position of this baseline may be specified in the baseline-table or it may be calculated.

NOTE: The position of this baseline is normally around or at the top of the ascenders, but it may not encompass all accents that can appear above a glyph. For these fonts the value of the "ascent" font characteristic is used. For ideographic fonts, the position of this baseline is normally 1 EM in the *shift-direction* from the "ideographic" baseline. However, some ideographic fonts have a reduced width in the inline-progression-direction to allow tighter setting. When such a font, designed only for vertical writing-modes, is used in a horizontal writing-mode, the "text-before-edge" baseline may be less than 1 EM from the text-after-edge.

text-after-edge

This identifies the after-edge of the EM box. The position of this baseline may be specified in the baseline-table or it may be calculated.

NOTE: For fonts with descenders, the position of this baseline is normally around or at the bottom of the descenders. For these fonts the value of the "descent" font characteristic is used. For ideographic fonts, the position of this baseline is normally at the "ideographic" baseline.

There are, in addition, two computed baselines that are only defined for line areas. For each line-area, there is a dominant-baseline, a baseline-table and a baseline-table font-size which are those of the nearest ancestor formatting object that completely contains the whole line. The "before-edge" and "after-edge" baselines are defined as follows.

before-edge

The offset of the "before-edge" baseline of the line from the dominant-baseline of the line is determined by ignoring all inline-areas whose alignment-baseline is either "before-edge" or "after-edge". For the "before-edge", extents are measured from the dominant-baseline in the direction toward the top of the reference-area. The top of the reference-area is defined by the reference-area's *reference-orientation*. The "before-edge" baseline offset is set to the maximum extent of the "before-edges" of the allocation-rectangles of the remaining areas. If all the inline-areas in a line-area are aligned either to the "before-edge" or to the "after-edge", then use the offset of the "text-before-edge" baseline of the line as the offset of the "before-edge" baseline of the line.

after-edge

The offset of the "after-edge" baseline of the line from the dominant-baseline of the line is determined by ignoring all inline-areas whose alignment-baseline is *after-edge*. For the "after-edge", extents are measured from the dominant-baseline in the direction toward the bottom of the reference-area. The top of the reference-area is defined by the reference-area's *reference-orientation*. The "after-edge" baseline offset is set to the negative of the maximum of

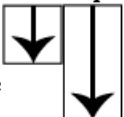



(1) the maximum extent of the "after-edges" of the allocation-rectangles of the remaining areas and (2) the maximum height of the allocation-rectangles of the areas that are ignored minus the offset of the "before-edge" baseline of the line.

NOTE: If all the inline-areas in a line-area are aligned to the "after-edge" then the specification for the "before-edge" will set the "before-edge" baseline to coincide with the "text-before-baseline" of the line. Then, case (2) above will determine an offset to the "bottom-edge" baseline that will align the "before-edge" of the area with the greatest height to its allocation-rectangle to "before-edge" baseline.

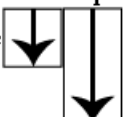
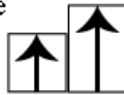

NOTE: The above specifications for "before-edge" and "after-edge" have the following three properties: (1) the allocation-rectangles of all the areas are below the "before-edge", (2) the allocation-rectangles of all the areas are above the "after-edge", and (3) the distance between the "before-edge" and the "after-edge" cannot be decreased without violating (1) or (2). The specified placement of the "before-edge" and "after-edge" is not the only way that (1)-(3) can be satisfied, but it is the only way they can be satisfied with the smallest possible offset to the "before-edge".

Examples showing "before-edge" and "after-edge" alignment:

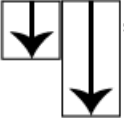

Ex 1: This is a longer line of text that provides a context for the line with the images aligned on it.

This is a line with before-edge , after-edge , middle , and no  alignment. And this is the line of text the follows that line with the aligned objects on it. And, finally, this line illustrates the normal spacing between lines of text in this font.

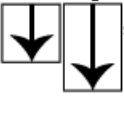
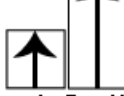
Ex 2: This is a longer line of text that provides a context for the line with the images aligned on it.

This is a line with before-edge , after-edge , middle  alignment. And this is the line of text the follows that line with the aligned objects on it. And, finally, this line illustrates the normal spacing between lines of text in this font.

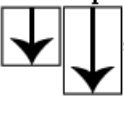
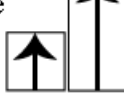

Ex 3: This is a longer line of text that provides a context for the line with the images aligned on it.

This is a line with before-edge , after-edge  alignment. And this is the line of text the follows that line with the aligned objects on it. And, finally, this line illustrates the normal spacing between lines of text in this font.

Ex 4: This is a longer line of text that provides a context for the line with the images aligned on it.

This is a line with before-edge , after-edge  alignment. And this is the line of text the follows that line with the aligned objects on it. And, finally, this line illustrates the normal spacing between lines of text in this font.

Ex 5: This is a longer line of text that provides a context for the line with the images aligned on it.

This is a line with before-edge , after-edge , middle  alignment. And this is the line of text the follows that line with the aligned objects on it. And, finally, this line illustrates the normal spacing between lines of text in this font.

Done

My Computer

The rectangles with lines or arrows are images with an intrinsic size as shown. The rectangles with no

arrows represent images that receive the default, dominant baseline, alignment. The alignment of the other rectangles is at the furthest point from the arrow head (which is in the middle when there are two arrowheads). Examples 1 and 2 show the "before-edge" alignment is determined by the tallest non-"before-edge" aligned objects: in example 1 this is the default aligned, arrowhead free rectangular image and in example 2 this is the double headed arrow rectangle. Examples 3 and 4 show defaulting to the "text-before-edge" when all the areas have either "before-edge" or "after-edge" alignment. In example 3, the images with "before-edge" alignment has a taller member than do the "after-edge" aligned images. In example 4, the tallest image is in the "after-edge" aligned set. Example 5 is a repetition of example 2 with largest image being an "after-edge" aligned image.

There are also four baselines that are defined only for horizontal writing-modes.

top

This baseline is the same as the "before-edge" baseline in a horizontal writing-mode and is undefined in a vertical writing mode.

text-top

This baseline is the same as the "text-before-edge" baseline in a horizontal writing-mode and is undefined in a vertical writing mode.

bottom

This baseline is the same as the "after-edge" baseline in a horizontal writing-mode and is undefined in a vertical writing mode.

text-bottom

This baseline is the same as the "text-after-edge" baseline in a horizontal writing-mode and is undefined in a vertical writing mode.

The alignment of a formatting object with respect to its parent is determined by three things: the scaled-baseline-table of the parent and the alignment-baseline and alignment-point of the formatting object being aligned. Prior to alignment, the scaled-baseline-table of the parent may be shifted. The property specifications below provide the information necessary to align the parent and child formatting objects.

There are four properties that control alignment of formatting objects to the above set of baselines. These properties are all independent and are designed so that typically only the specification of one of the properties is needed to achieve a particular alignment goal.

The primary baseline alignment property is the "dominant-baseline" property. This property has a compound value with three components. The dominant-baseline-identifier component is the default alignment-baseline to be used when aligning two inline areas. The baseline-table component specifies the positions of the baselines in the font design space coordinates. (See § 7.7.1 – [Fonts and Font Data](#) on page 190.) The baseline-table acts something like a musical staff; it defines particular points along the *block-progression-direction* to which glyphs and inline formatting objects can be aligned. The baseline-table font-size component provides a scaling factor for the baseline-table.

For convenience, the specification will sometimes refer to the baseline identified by the dominant-baseline-identifier component of the "dominant-baseline" property as the "dominant baseline" (in an abuse of terminology).

A simple example of alignment is shown in the following figure. The figure shows the presentation of two inline formatting objects, one inside the other. These inline formatting objects make up the content of a line in a block where the writing-mode is "lr-tb" and the font is "Helvetica". The structure of the example is as follows:

```
<fo:inline>Apex <fo:inline>Top</fo:inline></fo:inline>
```

Because no properties are specified, the initial values apply. Since a horizontal writing-mode is in use, the dominant-baseline-identifier is set to "alphabetic" and the baseline-table is taken from the nominal-font for the block in which the line appears, which, in this case, is Helvetica.

In the figure, the positions of the baselines relative to the current font size are shown as red (staff) lines. These lines are labeled with abbreviations of the names of the baselines (e.g., TBE for "text-before-edge"). The baseline identified by the dominant-baseline-identifier (A) is shown in blue. There is a break in the staff lines to separately show the inner inline formatting object. This is not necessary for this example, but this distinction will become important in subsequent examples.

The "alignment-baseline" property is the primary control on the positioning of an inner formatting object with respect to its parent. For all but fo:character, the initial value of the "alignment-baseline" property is "baseline". This aligns the dominant-baseline of the inner inline formatting object with the dominant baseline of the outer inline formatting object. This is shown by the short blue line that connects the two separated staffs (A) in the figure.

The glyphs determined by the fo:characters that are in the content of the two formatting objects are aligned based on the script to which the glyph belongs. Since this example only has Latin glyphs, they are aligned to the "alphabetic" baseline.



An inner inline formatting object containing Latin characters aligned to an outer inline formatting object also containing Latin characters.

In the next figure, the content of the inner inline formatting object is in Gurmukhi, the script of the Punjabi language. The Gurmukhi syllables are read as, "guru". Rather than use Unicode values for these characters, they are symbolized by placing the Latin transliteration in italic type. The structure of the example becomes:

```
<fo:inline>Apex <fo:inline>guru</fo:inline></fo:inline>
```

The only change from the previous example is that the glyphs of the Gurmukhi script are aligned to the "hanging" baseline of the inner inline formatting object. The alignment of that formatting object itself, with respect to the outer inline formatting object, is unchanged.



An inner inline formatting object containing Gurmukhi characters aligned to an outer inline formatting object containing Latin characters.

In the next figure, fragments of the text of the previous examples make up the content of the outer inline formatting object. The inner inline formatting object has a change of font-size, however. The structure is:

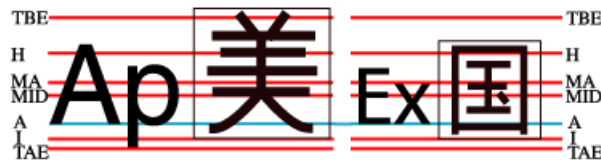
```
<fo:inline>Apguru
  <fo:inline font-size='.75em'>
    Exji
  </fo:inline>
</fo:inline>
```

In this example, the alignment of the inner inline formatting object itself does not change, nor does the alignment of the smaller glyphs inside the inner formatting object. The Latin glyphs are still aligned to the "alphabetic" baseline and the Gurmukhi glyphs, which are pronounced "ji" are aligned to the "hanging" baseline. Note also that just changing the "font-size" property did not change the baseline-table in effect in the inner inline formatting object.



The inner inline formatting object has a reduced font-size.

The next figure is equivalent to the previous example with the Gurmukhi character replaced by ideographic characters. These are aligned to the "ideographic" baseline.



The previous figure re-done with ideographic glyphs instead of Gurmukhi glyphs. The EM boxes are shown for the ideograms to clarify the alignment of these glyphs.

To change the scaling of the lines of the baseline table, it is necessary to use the "dominant-baseline" property on the inner inline formatting object. The value of "reset-size" causes the baseline-table font-size to be reset from the font-size of the formatting object on which the "dominant-baseline" property appears. The next figure shows the effect of this, using the structure:

```
<fo:inline>Apguru
  <fo:inline font-size='.75em'
    dominant-baseline='reset-size'>
    Exji
  </fo:inline>
</fo:inline>
```

The alignment of the inner inline formatting object, with respect to the outer inline formatting object, is still determined by aligning the dominant baselines. But, the baseline-table of the inner inline formatting object has been rescaled to the font-size of the inner inline formatting object. Hence the smaller glyphs align with each other.



The baseline-table of the inner inline formatting object has been re-sized to match the font-size of the inner inline formatting object.

But, what if it is more important that the small Gurmukhi glyphs align with the large Gurmukhi glyphs than having the Latin glyphs align. There are at least two ways to achieve this. The structure:

```
<fo:inline dominant-baseline='hanging'>Apguru
  <fo:inline font-size='.75em'
    dominant-baseline='reset-size'>
    Exji
  </fo:inline>
</fo:inline>
```

is illustrated in the next figure. The "hanging" baseline becomes the dominant baseline and the initial value of the "alignment-baseline" property causes the (newly) dominant "hanging" baselines to be aligned as is shown by the connection of the blue baselines.



Changing the dominant baseline to the "hanging" baseline causes the inner inline formatting object to be aligned on its parent's "hanging" baseline.

It is also possible to achieve the effect of the above figure without changing the dominant baseline. Instead it is sufficient to explicitly specify that the inner inline formatting object is aligned on its "hanging" baseline. This is done by:

```
<fo:inline>Apguru
  <fo:inline font-size='.75em'
    dominant-baseline='reset-size'
    alignment-baseline='hanging'>
    Exji
  </fo:inline>
</fo:inline>
```

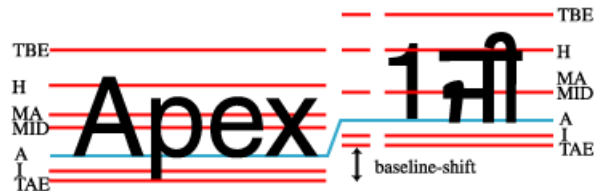
The only change this approach would make in the above figure is to color the "hanging" baseline red and keep the "alphabetic" baseline as the (blue) dominant baseline. This baseline in the inner inline formatting object would not (as it does not in the above figure) align with the "alphabetic" baseline in the outer inline formatting object.

The third baseline alignment property is the "baseline-shift" property. Like the properties other than the "dominant-baseline" property, this property does not change the baseline-table or the baseline-table font-size. It does shift the whole baseline table of the parent formatting object so that when an inner inline formatting object is aligned to one of the parents baselines, the position of the inner inline formatting object is shifted. This is illustrated in the next figure. The structure which creates this figure

is:

```
<fo:inline>Ap
  <fo:inline baseline-shift='super'>1ji</fo:inline>
</fo:inline>
```

Because the whole set of baseline-table staff lines are shifted to the position of the superscript baseline: it does not matter to which baseline the glyphs in the superscript are aligned. The European number "1" is aligned to the "alphabetic" baseline and the Gurmukhi syllable "ji" is aligned to the "hanging" baseline.

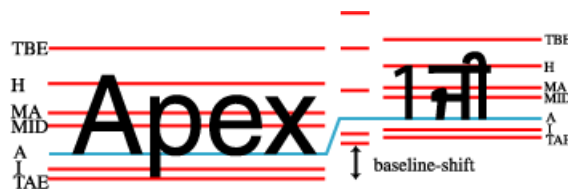


Using a "baseline-shift" for a super-script (or a sub-script).

It is more common for the font-size of the super-script text to be smaller than the font-size of the text to which it is appended. Consider:

```
<fo:inline>Ap
  <fo:inline font-size='.75em'
    baseline-shift='super'>
    1ji
  </fo:inline>
</fo:inline>
```

Because changing the font-size on a super-script (or sub-script) is common, this is the one case where changing the font-size does cause the baseline-table font-size to be reset when the "dominant-baseline" property has its initial value. After the rescaling, the default alignment to the dominant baseline positions the inline formatting object for the super-script to the dominant baseline position in the shifted baseline-table of the parent formatting object.



Reducing the font-size of the super-script automatically resets the baseline-table size so that mixed languages in the super-script stay mutually aligned.

The fourth alignment property is the "alignment-adjust" property. This property is primarily used for objects, such as some graphics, that do not belong to a particular script and do not have a predefined alignment point. The "alignment-adjust" property allows the author to assign where, on the start-edge of the object, the alignment point for that object lies.

7.11.1. “alignment-adjust”

XSL Definition:

<i>Value:</i>	auto baseline before-edge text-before-edge middle central after-edge text-after-edge ideographic alphabetic hanging mathematical <percentage> <length> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	all inline formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	see prose
<i>Media:</i>	visual

The "alignment-adjust" property allows more precise alignment of areas generated by formatting objects, in particular for formatting objects, such as graphics, that do not have a baseline-table or lack the desired baseline in their baseline-table. With the "alignment-adjust" property, the position of the baseline identified by the "alignment-baseline" can be explicitly determined.

Values for the property have the following meaning:

auto

For a glyph, the alignment-point is the intersection of the start-edge of the allocation-rectangle of the glyph-area and the block-progression-direction position of the alignment-point from the font as specified in § 7.7.1 – [Fonts and Font Data](#) on page 190. For other inline-areas, the alignment-point is at the intersection of the start-edge of the allocation-rectangle and the baseline identified by the "alignment-baseline" property if this baseline exists in the baseline-table for the dominant-baseline for the inline-area. If the baseline-identifier does not exist in the baseline-table for the glyph or other inline-area, then the User Agent may either use heuristics to determine where that missing baseline would be or may use the dominant-baseline as a fallback. For areas generated by a fo:external-graphic, or fo:instream-foreign-object, the alignment point is at the intersection of the start-edge and after-edge of the allocation-rectangle of the area.

baseline

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the dominant-baseline of the area.

before-edge

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "before-edge" baseline of the area.

text-before-edge

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "text-before-edge" baseline of the area.

central

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "central" baseline of the area.

middle

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "middle" baseline of the area.

after-edge

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "after-edge" baseline of the area.

text-after-edge

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "text-after-edge" baseline of the area.

ideographic

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "ideographic" baseline of the area.

alphabetic

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "alphabetic" baseline of the area.

hanging

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "hanging" baseline of the area.

mathematical

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "mathematical" baseline of the area.

top

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "top" baseline of the area if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

bottom

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "bottom" baseline of the area if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

text-top

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "text-top" baseline of the area if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

text-bottom

The alignment-point is at the intersection of the start-edge of the allocation-rectangle and the "text-bottom" baseline of the area if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

<percentage>

The computed value of the property is this percentage multiplied by the area's computed "height" if the area is generated by an `fo:external-graphic` or `fo:instream-foreign-object`, the "font-size" if the area was generated by an `fo:character`, and the "line-height" otherwise. The alignment-point is on the start-edge of the allocation-rectangle of the area being aligned. Its position along the start-edge relative to the intersection of the dominant-baseline and the start-edge is offset by the computed value. The offset is opposite to the *shift-direction* if that value is positive and in the *shift-direction* if that value is negative value). A value of "0%" makes the dominant-baseline the alignment point.

<length>

The alignment-point is on the start-edge of the allocation-rectangle of the area being aligned. Its position along the start-edge relative to the intersection of the dominant-baseline and the start-edge is offset by <length> value. The offset is opposite to the *shift-direction* if that value is positive and in the *shift-direction* if that value is negative. A value of "0cm" makes the dominant-baseline the alignment point.

7.11.2. "alignment-baseline"

XSL Definition:

<i>Value:</i>	auto baseline before-edge text-before-edge middle central after-edge text-after-edge ideographic alphabetic hanging mathematical inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	all inline formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property specifies how an object is aligned with respect to its parent. That is, to which of the parent's baselines the alignment-point of this object is aligned. The alignment-adjust property specifies how the alignment point is determined. It defaults to the baseline with the same name as the computed value of the alignment-baseline property. That is, the position of "ideographic" alignment-point in the *block-progression-direction* is the position of the "ideographic" baseline in the baseline-table of the object being aligned.

Values have the following meanings:

auto

The computed value depends on the kind of object on which it is being used. For `fo:character`, the value is the dominant-baseline of the script to which the character belongs. If the value of the "script" property on the parent formatting object is other than "auto" then use the baseline for that script; otherwise, use the dominant-baseline of the parent. For all other objects, the value is computed as for the "baseline" value.

baseline

The alignment-point of the object being aligned is aligned with the dominant-baseline of the parent area.

before-edge

The alignment-point of the object being aligned is aligned with the "before-edge" baseline of the parent area.

text-before-edge

The alignment-point of the object being aligned is aligned with the "text-before-edge" baseline of the parent area.

central

The alignment-point of the object being aligned is aligned with the "central" baseline of the parent area.

middle

The alignment-point of the object being aligned is aligned with the "middle" baseline of the parent area.

after-edge

The alignment-point of the object being aligned is aligned with the "after-edge" baseline of the parent area.

text-after-edge

The alignment-point of the object being aligned is aligned with the "text-after-edge" baseline of the parent area.

ideographic

The alignment-point of the object being aligned is aligned with the "ideographic" baseline of the parent area.

alphabetic

The alignment-point of the object being aligned is aligned with the "alphabetic" baseline of the parent area.

hanging

The alignment-point of the object being aligned is aligned with the "hanging" baseline of the parent area.

mathematical

The alignment-point of the object being aligned is aligned with the "mathematical" baseline of the parent area.

top

The alignment-point of the object being aligned is aligned with the "top" baseline of the parent area if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

bottom

The alignment-point of the object being aligned is aligned with the "bottom" baseline of the parent area if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

text-top

The alignment-point of the object being aligned is aligned with the "text-top" baseline of the parent area if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

text-bottom

The alignment-point of the object being aligned is aligned with the "text-bottom" baseline of the parent area if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

7.11.3. “baseline-shift”

XSL Definition:

<i>Value:</i>	baseline sub super <percentage> <length> inherit
<i>Initial:</i>	baseline
<i>Applies to:</i>	all inline formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	refers to the "line-height" of the parent area
<i>Media:</i>	visual

The "baseline-shift" property allows repositioning of the dominant-baseline relative to the dominant-baseline of the parent area. The shifted object might be a sub- or superscript. Within the shifted object, the whole baseline-table is offset; not just a single baseline. The amount of the shift is determined from information from the parent area, the sub- or superscript offset from the *nominal-font* of the parent area, percent of the "line-height" of the parent area or an absolute value.

When the value of "baseline-shift" is other than "0", then the baseline-table font-size component of the "dominant-baseline" property is re-computed to use the "font-size" applicable to the formatting object on which the non-zero "baseline-shift" property is specified.

Values for the property have the following meaning:

baseline

There is no baseline shift; the dominant-baseline remains in its original position.

sub

The dominant-baseline is shifted to the default position for subscripts. The offset to this position is determined by the font data for the nominal font as adjusted by the dominant baseline-table font-size. If there is no applicable font data the User Agent may use heuristics to determine the offset.

super

The dominant-baseline is shifted to the default position for superscripts. The offset to this position is determined by the font data for the nominal font as adjusted by the dominant baseline-table font-size. If there is no applicable font data the User Agent may use heuristics to determine the offset.

<percentage>

The computed value of the property is this percentage multiplied by the computed "line-height"

of the parent area. The dominant-baseline is shifted in the *shift-direction* (positive value) or opposite to the *shift-direction* (negative value) of the parent area by the computed value. A value of "0%" is equivalent to "baseline".

<length>

The dominant-baseline is shifted in the *shift-direction* (positive value) or opposite to the *shift-direction* (negative value) of the parent area by the <length> value. A value of "0cm" is equivalent to "baseline".

NOTE: Although it may seem that "baseline-shift" and "alignment-adjust" properties are doing the same thing, there is an important although, perhaps, subtle difference. For "alignment-adjust" the percentage values refer to the "line-height" of the area being aligned. For "baseline-shift" the percentage values refer to the "line-height" of the parent. Similarly, it is the "sub" and "super" offsets of the parent that are used to align the shifted baseline rather than the "sub" or "super" offsets of the areas being positioned. To ensure a consistent sub- or superscript position, it makes more sense to use the parent as the reference rather than the subscript formatting object which may have a changed "line-height" due to "font-size" changes in the sub- or superscript formatting object.

Using the "alignment-adjust" property is more suitable for positioning objects, such as graphics, that have no internal textual structure. Using the "baseline-shift" property is intended for sub- and superscripts where the positioned object may itself be a textual object. The baseline-shift provides a way to define a specific baseline offset other than the named offsets that are defined relative to the dominant-baseline. In addition, having "baseline-shift" makes it easier for tool to generate the relevant properties; many formatting programs already have a notion of baseline shift.

7.11.4. "display-align"

XSL Definition:

<i>Value:</i>	auto before center after inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:table-cell, fo:region-body, fo:region-before, fo:region-after, fo:region-start, fo:region-end
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property specifies the alignment, in the block-progression-direction, of the areas that are the children of a reference-area.

Values for the property have the following meaning:

auto

If the "relative-align" property applies to this formatting object the "relative-align" property is used. If not, this value is treated as if "before" had been specified.

before

The before-edge of the allocation-rectangle of the first child area is placed coincident with the before-edge of the content-rectangle of the reference-area.

center

The child areas are placed such that the distance between the before-edge of the allocation-rectangle of the first child area and the before-edge of the content-rectangle of the reference-area is the same as the distance between the after-edge of the allocation-rectangle of the last child area and the after-edge of the content-rectangle of the reference-area.

after

The after-edge of the allocation-rectangle of the last child area is placed coincident with the after-edge of the content-rectangle of the reference-area.

7.11.5. “dominant-baseline”

XSL Definition:

<i>Value:</i>	auto use-script no-change reset-size ideographic alphabetic hanging mathematical inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	all inline formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

The "dominant-baseline" property is used to determine or re-determine a scaled-baseline-table. A scaled-baseline-table is a compound value with three components: a baseline-identifier for the dominant-baseline, a baseline-table and a baseline-table font-size. Some values of the property re-determine all three values; other only re-establish the baseline-table font-size. When the initial value, "auto", would give an undesired result, this property can be used to explicitly set the desired scaled-baseline-table.

Values for the property have the following meaning:

auto

If this property occurs on a block-level formatting object, then the computed value depends on the value of the "script" property. There are two cases. If the value of the "script" property is "auto", then, if the "writing-mode" is horizontal, then the value of the dominant-baseline component is "alphabetic", else if the "writing-mode" is vertical, then the value of the dominant-baseline component is "central". On the other hand, if the value of the "script" property is anything other than "auto", then the value of the "script" property is used to select the dominant baseline. The "writing-mode", whether horizontal or vertical, is used to select the baseline-table that corresponds to that baseline. The baseline-table font-size component is set to the value of the "font-size" property on the formatting object on which the "dominant-baseline" property occurs.

Otherwise, if this property is not on a block-level formatting object, then the dominant-baseline and the baseline-table components remain the same as those of the parent formatting object. If the computed "baseline-shift" value actually shifts the baseline, then the baseline-table font-size component is set to the value of the "font-size" property on the formatting object on which the "dominant-baseline" property occurs, otherwise the baseline-table font-size remains the same as that of the parent formatting object. If there is no parent formatting object, the

scaled-baseline-table value is constructed as above for block-level formatting-objects.

use-script

The dominant-baseline and the baseline-table components are set using the value of the "script" property. The value of the "script" property is used to select the dominant baseline. The "writing-mode", whether horizontal or vertical, is used to select the appropriate set of baseline-tables and the dominant baseline is used to select the baseline-table that corresponds to that baseline. The baseline-table font-size component is set to the value of the "font-size" property on the formatting object on which the "dominant-baseline" property occurs.

no-change

The dominant-baseline, the baseline-table, and the baseline-table font-size remain the same as that of the parent formatting object.

reset-size

The dominant-baseline and the baseline-table remain the same, but the baseline-table font-size is changed to the value of the "font-size" property on this formatting object. This re-scales the baseline-table for the current "font-size".

ideographic

The dominant-baseline is set to the "ideographic" baseline using the baseline-table and baseline-table font-size of the parent area, the baseline-table is changed to correspond to the "ideographic" baseline, and the baseline-table font-size is changed to the value of the "font-size" property on this formatting object.

alphabetic

The dominant-baseline is set to the "alphabetic" baseline using the baseline-table and baseline-table font-size of the parent area, the baseline-table is changed to correspond to the "alphabetic" baseline, and the baseline-table font-size is changed to the value of the "font-size" property on this formatting object.

hanging

The dominant-baseline is set to the "hanging" baseline using the baseline-table and baseline-table font-size of the parent area, the baseline-table is changed to correspond to the "hanging" baseline, and the baseline-table font-size is changed to the value of the "font-size" property on this formatting object.

mathematical

The dominant-baseline is set to the "mathematical" baseline using the baseline-table and baseline-table font-size of the parent area, the baseline-table is changed to correspond to the "mathematical" baseline, and the baseline-table font-size is changed to the value of the "font-size" property on this formatting object.

If there is no baseline-table in the nominal font or if the baseline-table lacks an entry for the desired baseline, then the User Agent may use heuristics to determine the position of the desired baseline.

7.11.6. “relative-align”

XSL Definition:

<i>Value:</i>	before baseline inherit
<i>Initial:</i>	before
<i>Applies to:</i>	fo:list-item, fo:table-cell
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property specifies the alignment, in the block-progression-direction, between two or more areas. If the "display-align" property applies to this formatting object and has a value other than "auto" this property is ignored.

Values for the property have the following meaning:

before

For an fo:table-cell: for each row, the first child area of all the cells that start in the row and that have this value is placed such that the before-edge of the content-rectangle is placed at the same distance from the row grid. In addition, at least, one of these first child areas of the cells has to be placed with the before-edge of its allocation-rectangle coincident with the before-edge of the content-rectangle of the table-cell.

For an fo:list-item the before-edge of the first area descendant generated by the fo:list-item-label is placed coincident with the before-edge of the area generated by the fo:list-item. Similarly the before-edge of the first area descendant generated by the fo:list-item-body is placed coincident with the before-edge of the area generated by the fo:list-item.

baseline

For an fo:table-cell: for each row, the first child area of all the cells that start in the row and that have this value is placed such that the dominant-baseline, as specified on the fo:table-row, of the first line is placed at the same distance from the row grid. In addition, at least, one of these first child areas of the cells has to be placed with the before-edge of its allocation-rectangle coincident with the before-edge of the content-rectangle of the table-cell.

NOTE: That is, for all applicable cells the baseline of all the first lines are all aligned and placed the minimum distance down in the block-progression-direction. It should be noted that the start-edges of the content-rectangles of the cells need not align.

For an fo:list-item the distance between the baseline of the first line-area of the first area descendant generated by the fo:list-item-label is the same as the distance between the baseline of the first line-area of the first area descendant generated by the fo:list-item-body. In addition, at least, one of these first area descendants has to be placed such that the before-edge of its allocation-rectangle is coincident with the before-edge of the content-rectangle of the list-item.

7.12. Area Dimension Properties

7.12.1. "block-progression-dimension"

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	auto <length> <percentage> <length-range> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	see prose
<i>Media:</i>	visual

This property specifies the block-progression-dimension of the content-rectangle. The user may specify an explicit size (<length> or <percentage>) or a <length-range>, allowing the size to be adjusted by the formatter.

This property does not apply when the "line-height" property applies to the same dimension of the areas generated by this formatting object.

Values have the following meanings:

auto

No constraint is imposed by this property. The block-progression-dimension is determined by the formatter taking all other constraints into account.

Specifying block-progression-dimension="auto" will set:

- block-progression-dimension.minimum="auto"
- block-progression-dimension.optimum="auto"
- block-progression-dimension.maximum="auto"

<length>

Specifies a fixed block-progression-dimension.

Specifying block-progression-dimension=<length> will set:

- block-progression-dimension.minimum=<length>
- block-progression-dimension.optimum=<length>
- block-progression-dimension.maximum=<length>

<percentage>

Specifies a percentage block-progression-dimension. The percentage is calculated with respect to the corresponding dimension of the closest area ancestor that was generated by a block-level formatting object. If that dimension is not specified explicitly (i.e., it depends on content's block-progression-dimension), the value is interpreted as "auto".

Specifying block-progression-dimension=<percentage> will set:

- block-progression-dimension.minimum=<percentage>

- block-progression-dimension.optimum=<percentage>
- block-progression-dimension.maximum=<percentage>

<length-range>

Specifies the dimension as a length-range, consisting of:

- block-progression-dimension.optimum

This is the preferred dimension of the area created; if minimum and maximum are identical, the area is of a fixed dimension. If they are, respectively, smaller and larger than optimum, then the area may be adjusted in dimension within that range.

A value of "auto" may be specified for optimum, indicating that there is no preferred dimension, but that the intrinsic or resolved dimension of the area should be used. If minimum and/or maximum are not also auto, then the dimension shall be constrained between those limits.

- block-progression-dimension.minimum
- block-progression-dimension.maximum

A value of "auto" may be specified for block-progression-dimension.maximum. This indicates that there is no absolute maximum limit, and the object may be sized to its intrinsic size.

Negative values for block-progression-dimension.minimum, block-progression-dimension.optimum, and block-progression-dimension.maximum are invalid and are treated as if "0pt" had been specified.

If the computed value of block-progression-dimension.maximum is less than the computed value of block-progression-dimension.optimum, it is treated as if the value of block-progression-dimension.optimum had been specified.

If the computed value of block-progression-dimension.minimum is greater than the computed value of block-progression-dimension.optimum, it is treated as if the value of block-progression-dimension.optimum had been specified.

If the value of block-progression-dimension.optimum is "auto" and the computed value of block-progression-dimension.minimum is greater than the computed value of block-progression-dimension.maximum the block-progression-dimension.minimum is treated as if the value of block-progression-dimension.maximum had been specified.

7.12.2. “content-height”

XSL Definition:

<i>Value:</i>	auto scale-to-fit <length> <percentage> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:external-graphic, fo:instream-foreign-object
<i>Inherited:</i>	no
<i>Percentages:</i>	intrinsic height

Media: visual

Values have the following meanings:

auto

The content-height should be the intrinsic content-height.

scale-to-fit

A size of the content-height equal to the height of the viewport. This value implies a certain scaling factor to be applied onto the content.

<length>

An absolute size for the content-height. This value implies a certain scaling factor to be applied onto the content.

<percentage>

A percentage representing a scaling factor for the content-height.

Specifies the content-height of some object (e.g., an external graphic). If the value is a percentage, the value of this property is the percentage applied to the intrinsic height.

7.12.3. “content-width”

XSL Definition:

Value: auto | scale-to-fit | <length> | <percentage> | inherit

Initial: auto

Applies to: fo:external-graphic, fo:instream-foreign-object

Inherited: no

Percentages: intrinsic height

Media: visual

Values have the following meanings:

auto

The content-width should be the intrinsic content-width.

scale-to-fit

A size of the content-width equal to the height of the viewport. This value implies a certain scaling factor to be applied onto the content.

<length>

An absolute size for the content-width. This value implies a certain scaling factor to be applied onto the content.

<percentage>

A percentage representing a scaling factor for the content-width.

Specifies the content-width of some object (e.g., an external graphic). If the value is a percentage, the value of this property is the percentage applied to the intrinsic width.

7.12.4. “height”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> auto inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	all elements but non-replaced inline elements, table columns, and column groups
<i>Inherited:</i>	no
<i>Percentages:</i>	see prose
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visudet.html#propdef-height>.

This property specifies the content height of boxes generated by block-level and replaced elements.

This property does not apply to non-replaced inline-level elements. The height of a non-replaced inline element's boxes is given by the element's (possibly inherited) 'line-height' value.

Values have the following meanings:

auto

The height depends on the values of other properties.

<length>

Specifies a fixed height.

<percentage>

Specifies a percentage height. The percentage is calculated with respect to the height of the generated box's containing block. If the height of the containing block is not specified explicitly (i.e., it depends on content height), the value is interpreted like "auto".

Negative values for 'height' are illegal.

XSL modifications to the CSS definition:

In XSL, this property is mapped to either "inline-progression-dimension" or "block-progression-dimension", based on the applicable values of the "writing-mode" and "reference-orientation" properties. Details on the mapping are given in § 5 – Property Refinement / Resolution on page 39.

For a discussion of the "height" property in tables see: <http://www.w3.org/TR/REC-CSS2/tables.html>

7.12.5. “inline-progression-dimension”

Writing-mode Relative Equivalent of a CSS2 Property.

<i>Value:</i>	auto <length> <percentage> <length-range> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	see prose
<i>Media:</i>	visual

This property specifies the inline-progression-dimension of the content-rectangle. The user may specify an explicit size (<length> or <percentage>) or a <length-range>, allowing the size to be adjusted by the formatter.

This property does not apply when the "line-height" property applies to the same dimension of the areas generated by this formatting object.

Values have the following meanings:

auto

No constraint is imposed by this property. The block-progression-dimension is determined by the formatter taking all other constraints into account.

Specifying inline-progression-dimension=auto will set:

- inline-progression-dimension.minimum=auto
- inline-progression-dimension.optimum=auto
- inline-progression-dimension.maximum=auto

<length>

Specifies a fixed inline-progression-dimension.

Specifying inline-progression-dimension=<length> will set:

- inline-progression-dimension.minimum=<length>
- inline-progression-dimension.optimum=<length>
- inline-progression-dimension.maximum=<length>

<percentage>

Specifies a percentage inline-progression-dimension. The percentage is calculated with respect to the corresponding dimension of the closest area ancestor that was generated by a block-level formatting object. If that dimension is not specified explicitly (i.e., it depends on content's inline-progression-dimension), the value is interpreted as "auto".

Specifying inline-progression-dimension=<percentage> will set:

- inline-progression-dimension.minimum=<percentage>
- inline-progression-dimension.optimum=<percentage>

- `inline-progression-dimension.maximum=<percentage>`

<length-range>

Specifies the dimension as a length-range, consisting of:

- `inline-progression-dimension.optimum`

This is the preferred dimension of the area created, if minimum and maximum are identical, the area is of a fixed dimension. If they are, respectively, smaller and larger than optimum, then the area may be adjusted in dimension within that range.

A value of "auto" may be specified for optimum, indicating that there is no preferred dimension, but that the intrinsic or resolved dimension of the area should be used. If minimum and/or maximum are not also auto, then the dimension shall be constrained between those limits.

- `inline-progression-dimension.minimum`
- `inline-progression-dimension.maximum`

A value of "auto" may be specified for `inline-progression-dimension.maximum`. This indicates that there is no absolute maximum limit, and the object may be sized to its intrinsic size.

Negative values for `inline-progression-dimension.minimum`, `inline-progression-dimension.optimum`, and `inline-progression-dimension.maximum` are invalid and are treated as if "Opt" had been specified.

If the computed value of `inline-progression-dimension.maximum` is less than the computed value of `inline-progression-dimension.optimum`, it is treated as if the value of `inline-progression-dimension.optimum` had been specified.

If the computed value of `inline-progression-dimension.minimum` is greater than the computed value of `inline-progression-dimension.optimum`, it is treated as if the value of `inline-progression-dimension.optimum` had been specified.

7.12.6. "max-height"

CSS2 Definition:

<i>Value:</i>	<code><length></code> <code><percentage></code> none inherit
<i>Initial:</i>	Opt
<i>Applies to:</i>	all elements except non-replaced inline elements and table elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to height of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visudet.html#propdef-max-height>.

These two properties ["max-height" and "max-width"] allow authors to constrain box heights to a certain range. Values have the following meanings:

none

(Only on "max-height") No limit on the height of the box.

<length>

Specifies a fixed minimum or maximum computed height.

<percentage>

Specifies a percentage for determining the computed value. The percentage is calculated with respect to the height of the generated box's containing block. If the height of the containing block is not specified explicitly (i.e., it depends on content height), the percentage value is interpreted like "auto".

XSL modifications to the CSS definition:

In XSL, this property is mapped to either "inline-progression-dimension" or "block-progression-dimension", based on the applicable values of the "writing-mode" and "reference-orientation" properties. Details on the mapping are given in § 5 – Property Refinement / Resolution on page 39.

7.12.7. “max-width”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> none inherit
<i>Initial:</i>	none
<i>Applies to:</i>	all elements except non-replaced inline elements and table elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visudet.html#propdef-max-width>.

These two properties ["max-height" and "max-width"] allow authors to constrain box widths to a certain range. Values have the following meanings:

none

(Only on "max-width") No limit on the width of the box.

<length>

Specifies a fixed minimum or maximum computed width.

<percentage>

Specifies a percentage for determining the computed value. The percentage is calculated with respect to the width of the generated box's containing block.

XSL modifications to the CSS definition:

In XSL, this property is mapped to either "inline-progression-dimension" or "block-progression-dimension", based on the applicable values of the "writing-mode" and "reference-orientation" properties. Details on the mapping are given in § 5 – [Property Refinement / Resolution](#) on page 39.

7.12.8. “min-height”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> inherit
<i>Initial:</i>	Opt
<i>Applies to:</i>	all elements except non-replaced inline elements and table elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to height of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visudet.html#propdef-min-height>.

XSL modifications to the CSS definition:

In XSL, this property is mapped to either "inline-progression-dimension" or "block-progression-dimension", based on the applicable values of the "writing-mode" and "reference-orientation" properties. Details on the mapping are given in § 5 – [Property Refinement / Resolution](#) on page 39.

7.12.9. “min-width”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> inherit
<i>Initial:</i>	depends on UA
<i>Applies to:</i>	all elements except non-replaced inline elements and table elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visudet.html#propdef-min-width>.

XSL modifications to the CSS definition:

In XSL, this property is mapped to either "inline-progression-dimension" or "block-progression-dimension", based on the applicable values of the "writing-mode" and "reference-orientation" properties. Details on the mapping are given in § 5 – [Property Refinement / Resolution](#) on page 39.

7.12.10. “scaling”

XSL Definition:

<i>Value:</i>	uniform non-uniform inherit
<i>Initial:</i>	uniform
<i>Applies to:</i>	fo:external-graphic, fo:instream-foreign-object
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

uniform

Scaling should preserve the aspect ratio.

non-uniform

Scaling need not preserve the aspect ratio.

Specifies whether scaling needs to preserve the intrinsic aspect ratio.

7.12.11. “scaling-method”

XSL Definition:

<i>Value:</i>	auto integer-pixels resample-any-method inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:external-graphic, fo:instream-foreign-object
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

The User Agent is free to choose either resampling, integer scaling, or any other scaling method.

integer-pixels

The User Agent should scale the image such that each pixel in the original image is scaled to the nearest integer number of device-pixels that yields an image less-than-or-equal-to the image size derived from the content-height, content-width, and scaling properties.

resample-any-method

The User Agent should resample the supplied image to provide an image that fills the size derived from the content-height, content-width, and scaling properties. The user agent may use any sampling method.

This property is used to indicate a preference in the scaling/sizing tradeoff to be used when formatting bitmapped graphics.

NOTE: This is defined as a preference to allow the user agent the flexibility to adapt to device limitations and to accommodate over-constrained situations involving min/max dimensions and scale factors.

7.12.12. “width”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> auto inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	all elements but non-replaced inline elements, table-rows, and row groups
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visudet.html#propdef-width>.

This property specifies the content width of boxes generated by block-level and replaced elements.

This property does not apply to non-replaced inline-level elements. The width of a non-replaced inline element's boxes is that of the rendered content within them (before any relative offset of children). Recall that inline boxes flow into line boxes. The width of line boxes is given by their containing block, but may be shorted by the presence of floats.

The width of a replaced element's box is intrinsic and may be scaled by the user agent if the value of this property is different than 'auto'.

Values have the following meanings:

auto

The width depends on the values of other properties.

<length>

Specifies a fixed width.

<percentage>

Specifies a percentage width. The percentage is calculated with respect to the width of the generated box's containing block.

Negative values for "width" are illegal.

XSL modifications to the CSS definition:

In XSL, this property is mapped to either "inline-progression-dimension" or "block-progression-dimension", based on the applicable values of the "writing-mode" and "reference-orientation" properties. Details on the mapping are given in § 5 – [Property Refinement / Resolution](#) on page 39.

7.13. Block and Line-related Properties

7.13.1. “hyphenation-keep”

XSL Definition:

<i>Value:</i>	auto column page inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:block
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

No restriction applies. The word may be hyphenated at the end of any region.

column

Both parts of a hyphenated word shall lie within a single column.

page

Both parts of a hyphenated word shall lie within a single page.

Controls whether hyphenation can be performed on the last line that fits in a given reference-area.

7.13.2. “hyphenation-ladder-count”

XSL Definition:

<i>Value:</i>	no-limit <number> inherit
<i>Initial:</i>	no-limit
<i>Applies to:</i>	fo:block
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

no-limit

Any number of successive lines may be hyphenated.

<integer>

An integer greater than or equal to 1.

Specifies a limit on the number of successive hyphenated line-areas the formatter may generate in a block-area.

7.13.3. “last-line-end-indent”

XSL Definition:

<i>Value:</i>	<length> <percentage> inherit
<i>Initial:</i>	0pt
<i>Applies to:</i>	fo:block
<i>Inherited:</i>	yes
<i>Percentages:</i>	width of containing block
<i>Media:</i>	visual

Values have the following meanings:

<length>

The "last-line-end-indent" is specified as a length.

<percentage>

The "last-line-end-indent" is specified as a percentage of the block's content-rectangle inline-progression-dimension.

Specifies an indent to be applied to the end-edge of the last (or only) line-area contained in a block-area. It is added to the block's end-edge. Positive values indent the edge, negative values outdent the edge.

7.13.4. “line-height”

CSS2 Definition:

<i>Value:</i>	normal <length> <number> <percentage> <space> inherit
<i>Initial:</i>	normal
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	refer to the font size of the element itself
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visudet.html#propdef-line-height>.

Values have the following meanings:

normal

Tells user agents to set the computed value to a "reasonable" value based on the font size of the element. The value has the same meaning as <number>. We recommend a computed value for "normal" between 1.0 to 1.2.

<length>

The box height is set to this length. Negative values are illegal.

<number>

The computed value of the property is this number multiplied by the element's font size. Negative values are illegal. However, the number, not the computed value, is inherited.

<percentage>

The computed value of the property is this percentage multiplied by the element's computed font size. Negative values are illegal.

If the property is set on a block-level element whose content is composed of inline-level elements, it specifies the minimal height of each generated inline box.

If the property is set on an inline-level element, it specifies the exact height of each box generated by the element. (Except for inline replaced elements, where the height of the box is given by the "height" property.)

When an element contains text that is rendered in more than one font, user agents should determine the "line-height" value according to the largest font size.

Generally, when there is only one value of "line-height" for all inline boxes in a paragraph (and no tall images), the above will ensure that baselines of successive lines are exactly "line-height" apart. This is important when columns of text in different fonts have to be aligned, for example in a table.

Note that replaced elements have a "font-size" and a "line-height" property, even if they are not used directly to determine the height of the box. The "font-size" is, however, used to define the "em" and "ex" units, and the "line-height" has a role in the "vertical-align" property.

XSL modifications to the CSS definition:

XSL adds the following value with the following meanings:

<space>

The difference between the inline-area's actual height and the line-height's space-specifier's three lengths are each divided by 2.0 and the result is used to set three half-leading values (optimum, minimum, and maximum).

Negative values for line-height.minimum, line-height.optimum, and line-height.maximum are invalid and will be interpreted as Opt.

If the value of line-height.maximum is less than the value of line-height.optimum, it is treated as if the value of line-height.optimum had been specified. If the value of line-height.minimum is greater than the value of line-height.optimum, it is treated as if the value of line-height.optimum had been specified.

The line-height.conditionality setting can be used to control the half-leading above the first line or after the last line that is placed in a reference-area.

The line-height.precedence setting can be used to control the merging of the half-leading with other spaces.

The *space-before* and *space-after* space-specifiers are set to the value of the half-leading. A definition of space-specifiers, and the interaction between space-specifiers occurring in sequence are given in [§ 4.3 – Spaces and Conditionality](#) on page 25.

If line-height is specified using <length>, <percentage>, or <number>, the formatter shall convert the single value to a space-specifier with the subfields interpreted as follows:

- line-height.minimum: the resultant computed value (as a length) of the <length>, <percentage>, or <number>.
- line-height.optimum: the resultant computed value (as a length) of the <length>, <percentage>, or <number>.
- line-height.maximum: the resultant computed value (as a length) of the <length>, <percentage>, or <number>.
- line-height.precedence: force.
- line-height.conditionality: retain

7.13.5. “line-height-shift-adjustment”

XSL Definition:

<i>Value:</i>	consider-shifts disregard-shifts inherit
<i>Initial:</i>	consider-shifts
<i>Applies to:</i>	fo:block
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

consider-shifts

In determining the line-height, include the adjusted top-edge and bottom-edge of any characters that have a baseline-shift.

disregard-shifts

In determining the line-height, include the unshifted top-edge and bottom-edge of any characters that have a baseline-shift.

This property is used to control whether the line-height is adjusted for content that has a baseline-shift.

NOTE: This property can be used to prevent superscript and subscript characters from disrupting the line-spacing.

7.13.6. “line-stacking-strategy”

XSL Definition:

<i>Value:</i>	line-height font-height max-height inherit
<i>Initial:</i>	line-height
<i>Applies to:</i>	fo:block
<i>Inherited:</i>	yes

Percentages: N/A
Media: visual

Values have the following meanings:

line-height

Matches CSS's line-height and positioning strategy. (Uses the per-inline-height-rectangle as described in the area model.)

font-height

Uses the block's font height as adjusted by the font-height-override-before and font-height-override-after properties. (Uses the nominal-requested-line-rectangle as described in the area model.)

max-height

Uses the adjusted maximum ascender-heights and maximum descender-depth for the actual fonts and inline-areas placed on the line. This value may be further influenced by the line-height-shift-adjustment property. (Uses the maximal-line-rectangle as described in the area model).

Selects the strategy for positioning adjacent lines, relative to each other.

7.13.7. “linefeed-treatment”

XSL Definition:

Value: ignore | preserve | treat-as-space | inherit
Initial: treat-as-space
Applies to: see prose
Inherited: yes
Percentages: N/A
Media: visual

Values have the following meanings:

ignore

Specifies that any character flow object whose Unicode character code is #xA shall be discarded.

preserve

Specifies no special action.

treat-as-space

Specifies that any character flow object whose Unicode character code is #xA shall be treated by subsequent XSL processing (including collapsing) and the formatter as if its Unicode character code were #x20.

treat-as-zero-width-space

Specifies that any character flow object whose Unicode character code is #xA shall be treated by subsequent XSL processing as if its Unicode character code were #x200B.

NOTE: The Unicode Standard recommends that the zero width space is considered a valid line-break point and that if two characters with a zero width space in between are placed on the same line they are placed with no space between them and that if they are placed on two lines no additional glyph area, such as for a hyphen, is created at the line-break.

The "linefeed-treatment" property specifies the treatment of linefeeds (#xA characters).

7.13.8. "space-treatment"

XSL Definition:

<i>Value:</i>	ignore preserve inherit
<i>Initial:</i>	preserve
<i>Applies to:</i>	see prose
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

ignore

Specifies that any character flow object whose character is classified as "white space" in XML, except for #xA (linefeed) characters (since their treatment is determine by the linefeed-treatment property), shall be discarded.

preserve

Specifies no special action.

ignore-if-before-linefeed

Specifies that any character flow object, or any sequence of character flow objects, whose character is classified as "white space" in XML, except for #xA (linefeed) characters (since their treatment is determine by the linefeed-treatment property), that immediately precedes a linefeed character, shall be discarded. This action shall take place regardless of the setting of the linefeed-treatment property.

ignore-if-after-linefeed

Specifies that any character flow object, or any sequence of character flow objects, whose character is classified as "white space" in XML, except for #xA (linefeed) characters (since their treatment is determine by the linefeed-treatment property), that immediately follows a linefeed character, shall be discarded. This action shall take place regardless of the setting of the linefeed-treatment property.

ignore-if-surrounding-linefeed

Specifies that any character flow object, or any sequence of character flow objects, whose

character is classified as "white space" in XML, except for #xA (linefeed) characters (since their treatment is determined by the linefeed-treatment property), that immediately precedes or follows a linefeed character, shall be discarded. This action shall take place regardless of the setting of the linefeed-treatment property.

The "space-treatment" property specifies the treatment of space (#x20) and other whitespace characters except for linefeeds (#xA characters).

7.13.9. "text-align"

CSS2 Definition:

<i>Value:</i>	start center end justify inside outside left right <string> inherit
<i>Initial:</i>	start
<i>Applies to:</i>	block-level elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/text.html#propdef-text-align>.

This property describes how inline content of a block is aligned. Values have the following meanings:

left

center

right

justify

Left, right, center, and double justify text, respectively.

<*string*>

Specifies a string on which cells in a table column will align (see the section on horizontal alignment in a column for details and an example). This value applies only to table cells. If set on other elements, it will be treated as 'left' or 'right', depending on whether 'direction' is 'ltr', or 'rtl', respectively.

A block of text is a stack of line boxes. In the case of 'left', 'right' and 'center', this property specifies how the inline boxes within each line box align with respect to the line box's left and right sides; alignment is not with respect to the viewport. In the case of 'justify', the UA may stretch the inline boxes in addition to adjusting their positions. (See also 'letter-spacing' and 'word-spacing'.)

NOTE: The actual justification algorithm used is user agent and written language dependent.

Conforming user agents may interpret the value 'justify' as 'left' or 'right', depending on whether the element's default writing direction is left-to-right or right-to-left, respectively.

XSL modifications to the CSS definition:

Values have the following meanings:

start

Specifies that the content is to be aligned on the start-edge in the inline-progression-direction.

center

Specifies that the content is to be centered in the inline-progression-direction.

end

Specifies that the content is to be aligned on the end-edge in the inline-progression-direction.

justify

Specifies that the contents is to be expanded to fill the available width in the inline-progression-direction.

The last (or only) line of any block will be aligned in accordance with the "text-align-last" property value. If this line is to be justified specify "text-align-last='justify'".

inside

If the page binding edge is on the start-edge, the alignment will be start. If the binding is the end-edge, the alignment will be end. If neither, use start alignment.

outside

If the page binding edge is on the start-edge, the alignment will be end. If the binding is the end-edge, the alignment will be start. If neither, use end alignment.

left

Interpreted as "text-align='start'".

right

Interpreted as "text-align='end'".

<string>

Specifies a string on which cells in a table column will align (see the section, in the CSS2 recommendation, on horizontal alignment in a column for details and an example). This value applies only to table cells. If set on other elements, it will be treated as "start".

This property describes how inline content of a block is aligned.

7.13.10. "text-align-last"

XSL Definition:

<i>Value:</i>	relative start center end justify inside outside left right inherit
<i>Initial:</i>	relative
<i>Applies to:</i>	fo:block

<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

relative

If text-align is justify, then the alignment of the last line will be start. If text-align is not justify, text-align-last will use the value of text-align.

start

Specifies that the content is to be aligned on the start-edge in the inline-progression-direction.

center

Specifies that the contents is to be centered in the inline-progression-direction.

end

Specifies that the content is to be aligned on the end-edge in the inline-progression-direction.

justify

Specifies that the contents is to be expanded to fill the available width in the inline-progression-direction.

inside

If the page binding edge is on the start-edge, the alignment will be start. If the binding is the end-edge, the alignment will be end. If neither, use start-side.

outside

If the page binding edge is on the start-edge, the alignment will be end. If the binding is the end-edge the alignment will be start. If neither, use end alignment.

left

Interpreted as "text-align-last='start'".

right

Interpreted as "text-align-last='end'".

Specifies the alignment of the last line-area in a block.

7.13.11. “text-indent”

CSS2 Definition:

<i>Value:</i>	<length> <percentage> inherit
<i>Initial:</i>	Opt
<i>Applies to:</i>	block-level elements

<i>Inherited:</i>	yes
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/text.html#propdef-text-indent>.

This property specifies the indentation of the first line of text in a block. More precisely, it specifies the indentation of the first box that flows into the block's first line box. The box is indented with respect to the left (or right, for right-to-left layout) edge of the line box. User agents should render this indentation as blank space.

Values have the following meanings:

<length>

The indentation is a fixed length.

<percentage>

The indentation is a percentage of the containing block width

The value of 'text-indent' may be negative, but there may be implementation-specific limits.

XSL modifications to the CSS definition:

The "text-indent" property specifies an adjustment to the start-indent of the first line-area in a block-area. This indent is added to the block's start-indent.

A negative value specifies a hanging indent (outdent) on the first line.

7.13.12. “white-space-collapse”

XSL Definition:

<i>Value:</i>	false true inherit
<i>Initial:</i>	true
<i>Applies to:</i>	see prose
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

false

Specifies no special action.

true

Specifies, for any character flow object such that:

- its character is classified as "white space" in XML, *and*
- it is *not* a preserved linefeed (due to `linefeed-treatment="preserve"`), *and*

- the immediately preceding (non-ignored) flow object was a character flow object also with a character of class white-space *or* the immediately *following* (non-ignored) flow object is a preserved linefeed,
- that flow object shall be ignored.

The "white-space-collapse" property specifies the treatment of consecutive white-space. The effect is as follows: after all ignored white-space is discarded and all "treat-as-space" white-space is turned into a space, all resulting runs of two or more consecutive spaces are replaced by a single space, then any remaining space immediately adjacent to a remaining linefeed is also discarded. An implementation is free to use any algorithm to achieve an equivalent effect.

7.13.13. "wrap-option"

XSL Definition:

<i>Value:</i>	no-wrap wrap inherit
<i>Initial:</i>	wrap
<i>Applies to:</i>	see prose
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

nowrap

No line-wrapping will be performed.

In the case when lines are longer than the available width of the content-rectangle, the overflow will be treated in accordance with the "overflow" property specified on the reference-area.

wrap

Line-breaking will occur if the line overflows the available block width. No special markers or other treatment will occur.

Specifies how line-wrapping (line-breaking) of the content of the formatting object is to be handled.

7.14. Character Properties

7.14.1. "character"

XSL Definition:

<i>Value:</i>	<character>
<i>Initial:</i>	N/A, value is required
<i>Applies to:</i>	fo:character
<i>Inherited:</i>	no, a value is required
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<character>

Specifies the Unicode character to be presented.

7.14.2. “letter-spacing”

CSS2 Definition:

Value: normal | <length> | <space> | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/text.html#propdef-letter-spacing>.

This property specifies spacing behavior between text characters. Values have the following meanings:

normal

The spacing is the normal spacing for the current font. This value allows the user agent to alter the space between characters in order to justify text.

<length>

This value indicates inter-character space in addition to the default space between characters. Values may be negative, but there may be implementation-specific limits. User agents may not further increase or decrease the inter-character space in order to justify text.

Character-spacing algorithms are user agent dependent. Character spacing may also be influenced by justification (see the "text-align" property).

When the resultant space between two characters is not the same as the default space, user agents should not use ligatures.

Conforming user agents may consider the value of the 'letter-spacing' property to be 'normal'.

XSL modifications to the CSS definition:

The following value type has been added for XSL:

<space>

This allows the user to specify a range of adjustments in addition to the default space between characters.

The minimum and maximum values specify the limits of the adjustment.

Default space between characters is defined to be 0pt, i.e., glyph-areas stacked with no extra space between the allocation-rectangles of the glyph-areas. The *inline-progression-dimension* of the

glyph-area is obtained by formatting the fo:character.

For an fo:character that in the Unicode database is classified as "Alphabetic" the *start-space* and *end-space* traits are each set to a value as follows:

- For "normal": .optimum = "the normal spacing for the current font" / 2, .maximum = auto, .minimum = auto, .precedence = force, and .conditionality = discard. A value of auto for a component implies that the limits are User Agent specific.
- For a <length>: .optimum = <length> / 2, .maximum = .optimum, .minimum = .optimum, .precedence = force, and .conditionality = discard.
- For a <space>: a value that is half the value of the "letter-spacing" property for the numeric components and the value for the .precedence and .conditionality components. The initial values for .precedence is "force" and for .conditionality "discard".

The CSS statement that "Conforming user agents may consider the value of the 'letter-spacing' property to be 'normal'." does not apply in XSL, if the User Agent implements the "Extended" property set.

NOTE: If it is desired that the letter-space combine with other spaces that have less than forcing precedence, then the value of the letter-space should be specified as a <space> with precedence less than force which implies that space combines according to the space resolution rules described in § 4.3 – Spaces and Conditionality on page 25.

The algorithm for resolving the adjusted values between word-spacing and letter-spacing is User Agent dependent.

7.14.3. “suppress-at-line-break”

XSL Definition:

<i>Value:</i>	auto suppress retain inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:character
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property applies only to fo:character and determines whether the character's representation shall be suppressed when it would occur adjacent to a formatter-generated line break. Multiple characters may be so suppressed.

This property has the following values:

auto

The value of this property is determined by the Unicode value of the object's character property. The character at codepoint u+0020 is treated as if 'suppress' had been specified. All other characters are treated as if 'retain' had been specified.

This value does not automatically suppress the presentation of the non-breaking-space (u+00a0), the fixed spaces (u+2000 through u+200a), or the ideographic-space (u+3000).

suppress

If the glyph-area generated by the fo:character is first or last in a line-building partition (see

section § 4.7.2 – [Line-building](#) on page 33) then it is deleted rather than being placed in the area tree, together with all adjacent areas with a `suppress-at-line-break` value of `suppress`. Otherwise it is retained and placed in the area tree.

retain

The glyph-area generated by the `fo:character` shall be placed in the area tree whether or not it is first or last in a line-building partition.

7.14.4. “text-decoration”

CSS2 Definition:

<i>Value:</i>	<code>none</code> [[<code>underline</code> <code>no-underline</code>] [<code>overline</code> <code>no-overline</code>] [<code>line-through</code> <code>no-line-through</code>] [<code>blink</code> <code>no-blink</code>]] <code>inherit</code>
<i>Initial:</i>	<code>none</code>
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no, but see prose
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/text.html#propdef-text-decoration>.

This property describes decorations that are added to the text of an element. If the property is specified for a block-level element, it affects all inline-level descendants of the element. If it is specified for (or affects) an inline-level element, it affects all boxes generated by the element. If the element has no content or no text content (e.g., the `IMG` element in HTML), user agents must ignore this property.

Values have the following meanings:

none

Produces no text decoration.

underline

Each line of text is underlined.

overline

Each line of text has a line above it.

line-through

Each line of text has a line through the middle

blink

Text blinks (alternates between visible and invisible). Conforming user agents are not required to support this value.

The color(s) required for the text decoration should be derived from the "color" property value.

This property is not inherited, but descendant boxes of a block box should be formatted with the same decoration (e.g., they should all be underlined). The color of decorations should remain the same even if descendant elements have different "color" values.

XSL modifications to the CSS definition:

XSL adds the following values with the following meanings:

no-underline

Turns off underlining, if any.

no-overline

Turns off overlining, if any.

no-line-through

Turns off line-through, if any.

no-blink

Turns off blinking, if any.

7.14.5. “text-shadow”

CSS2 Definition:

<i>Value:</i>	none [<color> <length> <length> <length>? ,]* [<color> <length> <length> <length>?] inherit
<i>Initial:</i>	none
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no, see prose
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/text.html#propdef-text-shadow>.

This property accepts a comma-separated list of shadow effects to be applied to the text of the element. The shadow effects are applied in the order specified and may thus overlay each other, but they will never overlay the text itself. Shadow effects do not alter the size of a box, but may extend beyond its boundaries. The stack level of the shadow effects is the same as for the element itself.

Each shadow effect must specify a shadow offset and may optionally specify a blur radius and a shadow color.

A shadow offset is specified with two "length" values that indicate the distance from the text. The first length value specifies the horizontal distance to the right of the text. A negative horizontal length value places the shadow to the left of the text. The second length value specifies the vertical distance below the text. A negative vertical length value places the shadow above the text.

A blur radius may optionally be specified after the shadow offset. The blur radius is a length value

that indicates the boundaries of the blur effect. The exact algorithm for computing the blur effect is not specified.

A color value may optionally be specified before or after the length values of the shadow effect. The color value will be used as the basis for the shadow effect. If no color is specified, the value of the "color" property will be used instead.

7.14.6. “text-transform”

CSS2 Definition:

<i>Value:</i>	capitalize uppercase lowercase none
<i>Initial:</i>	none
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/text.html#propdef-text-transform>.

This property controls capitalization effects of an element's text. Values have the following meanings:

capitalize

Puts the first character of each word in uppercase.

uppercase

Puts all characters of each word in uppercase.

lowercase

Puts all characters of each word in lowercase.

none

No capitalization effects.

The actual transformation in each case is written language dependent. See [RFC2070] for ways to find the language of an element.

Conforming user agents may consider the value of "text-transform" to be "none" for characters that are not from the ISO Latin-1 repertoire and for elements in languages for which the transformation is different from that specified by the case-conversion tables of Unicode or ISO 10646.

XSL modifications to the CSS definition:

There are severe internationalization issues with the use of this property. It has been retained for CSS compatibility, but its use is not recommended in XSL.

7.14.7. “treat-as-word-space”

XSL Definition:

<i>Value:</i>	auto yes no inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:character
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property determines if the character shall be treated as a word-space or as a normal letter.

This property has the following values:

auto

The value of this property is determined by the Unicode codepoint for the character.

As the default behavior:

- The characters at codepoints u+0020 and u+00a0 are treated as if 'yes' had been specified. All other characters are treated as if 'no' had been specified.
- This property does not automatically apply word-spacing to the fixed spaces (u+2000 through u+200a) or the ideographic-space (u+3000).
- This default behavior can be overridden by information in the font used for formatting the character, which can specify additional characters that may be treated as "word-spaces".

yes

This inline-progression-dimension of the character shall be adjusted as described in the "word-spacing" property.

no

This character shall not have a word-spacing adjustment applied.

7.14.8. “word-spacing”

CSS2 Definition:

<i>Value:</i>	normal <length> <space> inherit
<i>Initial:</i>	normal
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/text.html#propdef-word-spacing>.

This property specifies spacing behavior between words. Values have the following meanings:

normal

The normal inter-word space, as defined by the current font and/or the UA.

<length>

This value indicates inter-word space in addition to the default space between words. Values may be negative, but there may be implementation-specific limits.

Word spacing algorithms are user agent-dependent. Word spacing is also influenced by justification (see the 'text-align' property).

XSL modifications to the CSS definition:

The following value type has been added for XSL:

<space>

This allows the user to specify a range of adjustments in addition to the default space between words.

The minimum and maximum values specify the limits of the adjustment.

Default space between words is defined to be the *inline-progression-dimension* of the glyph-area obtained by formatting the current fo:character whose *treat-as-wordspace* trait has the value "yes".

For fo:character whose *treat-as-word-space* trait has the value "yes", the *start-space* and *end-space* traits are each set to a value as follows:

- For "normal": .optimum = ("the normal inter-word-space, as defined by the current font and/or the UA" - "the inline-progression-dimension of the glyph-area obtained by formatting the fo:character") / 2, .maximum = .optimum, .minimum = .optimum, .precedence = force, and .conditionality = discard.
- For a <length>: .optimum = <length> / 2, .maximum = .optimum, .minimum = .optimum, .precedence = force, and .conditionality = discard.
- For a <space>: a value that is half the value of the "word-spacing" property for the numeric components and the value for the .precedence and .conditionality components. The initial values for .precedence is "force" and for .conditionality "discard".

NOTE: If it is desired that the word-space combine with other spaces that have less than forcing precedence, then the value of the word-space should be specified as a <space> with precedence less than force which implies that space combines according to the space resolution rules described in § 4.3 – Spaces and Conditionality on page 25.

The algorithm for resolving the adjusted values between word-spacing and letter-spacing is User Agent dependent.

NOTE: The "word-spacing" property only affects the placement of glyphs and not the shape that may be associated with the characters. For example, adjusting a "_" treated as a word-space does not lengthen or shorten the "_" glyph.

7.15. Color-related Properties

7.15.1. “color”

CSS2 Definition:

<i>Value:</i>	<color> inherit
<i>Initial:</i>	depends on user agent
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/colors.html#propdef-color>.

<color>

Any valid color specification.

This property describes the foreground color of an element's text content.

XSL modifications to the CSS definition:

XSL adds an "icc-color" function (see § 5.10.2 – Color Functions on page 60) as a valid value of this property.

7.15.2. “color-profile-name”

XSL Definition:

<i>Value:</i>	<name> inherit
<i>Initial:</i>	
<i>Applies to:</i>	fo:color-profile
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

<name>

Specifies the name of a color-profile for internal references.

7.15.3. “rendering-intent”

XSL Definition:

<i>Value:</i>	auto perceptual relative-colorimetric saturation absolute-colorimetric inherit
<i>Initial:</i>	auto

<i>Applies to:</i>	fo:color-profile
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

"rendering-intent" permits the specification of a color-profile rendering-intent other than the default. "rendering-intent" is applicable primarily to color-profiles corresponding to CMYK color spaces. The different options cause different methods to be used for translating colors to the color gamut of the target rendering device.

Values have the following meanings:

auto

This is the default behavior. The user-agent determines the best intent based on the content type. For image content containing an embedded profile, it shall be assumed that the intent specified within the profile is the desired intent. Otherwise, the user agent shall use the current profile and force the intent, overriding any intent that might be stored in the profile itself.

perceptual

This method, often the preferred choice for images, preserves the relationship between colors. It attempts to maintain relative color values among the pixels as they are mapped to the target device gamut. Sometimes pixel values that were originally within the target device gamut are changed in order to avoid hue shifts and discontinuities and to preserve as much as possible the overall appearance of the scene.

relative-colorimetric

Leaves colors that fall inside the gamut unchanged. This method usually converts out of gamut colors to colors that have the same lightness but fall just inside the gamut.

saturation

Preserves the relative saturation (chroma) values of the original pixels. Out of gamut colors are converted to colors that have the same saturation but fall just inside the gamut.

absolute-colorimetric

Disables white point matching when converting colors. This option is generally not recommended.

7.16. Float-related Properties

7.16.1. "clear"

CSS2 Definition:

<i>Value:</i>	start end left right both none inherit
<i>Initial:</i>	none
<i>Applies to:</i>	block-level elements

<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-clear>.

This property indicates which sides of an element's box(es) may not be adjacent to an earlier floating box. (It may be that the element itself has floating descendants; the 'clear' property has no effect on those.)

This property may only be specified for block-level elements (including floats). For compact and run-in boxes, this property applies to the final block box to which the compact or run-in box belongs.

Values have the following meanings when applied to non-floating block boxes:

left

The top margin of the generated box is increased enough that the top border edge is below the bottom outer edge of any left-floating boxes that resulted from elements earlier in the source document.

right

The top margin of the generated box is increased enough that the top border edge is below the bottom outer edge of any right-floating boxes that resulted from elements earlier in the source document.

both

The generated box is moved below all floating boxes of earlier elements in the source document.

none

No constraint on the box's position with respect to floats.

When the property is set on floating elements, it results in a modification of the rules for positioning the float. An extra constraint (#10) is added [to those specified in the description of the 'float' property]:

10. The top outer edge of the float must be below the bottom outer edge of all earlier left-floating boxes (in the case of 'clear: left'), or all earlier right-floating boxes (in the case of 'clear: right'), or both ('clear: both').

XSL modifications to the CSS definition:

A start-float is defined to mean an area with area-class "xsl-side-float" that was generated by an fo:float with property "float" specified as "left" or "start".

An end-float is defined to mean an area with area-class "xsl-side-float" that was generated by an fo:float with property "float" specified as "left" or "start".

A side-float is defined to mean either a start-float or an end-float.

An area is defined to "clear" a side-float if the before-edge of the area's border-rectangle is positioned to be after the after-edge of the float, or if the area is not a descendant of the side-float's parent reference-area.

A block-level formatting object is defined "to clear" a side-float if the areas generated by the formatting object clear the side-float.

In XSL this property applies to block-level formatting objects and fo:float.

The clear property when applied to an fo:float that generates side-floats does not apply to the fo:float's anchor-area.

Values have the following meanings:

start

Specifies that each area generated by the formatting object must clear every start-float whose parent reference-area is the nearest ancestor reference-area of the generated area, provided the start-float was generated by an fo:float that is before this formatting object, using pre-order traversal order of the formatting tree. Additionally specifies that each area generated by the formatting object must be placed so that the reference-area chain containing the generated area's nearest ancestor reference-area does not contain a later reference-area that is the parent of a start-float generated by an fo:float that is before this formatting object, using pre-order traversal of the formatting tree.

end

Specifies that each area generated by the formatting object must clear every end-float whose parent reference-area is the nearest ancestor reference-area of the generated area, provided the end-float was generated by an fo:float that is before this formatting object, using pre-order traversal order of the formatting tree. Additionally specifies that each area generated by the formatting object must be placed so that the reference-area chain containing the generated area's nearest ancestor reference-area does not contain a later reference-area that is the parent of an end-float generated by an fo:float that is before this formatting object, using pre-order traversal of the formatting tree.

left

Interpreted as "clear='start'".

right

Interpreted as "clear='end'".

both

Specifies that each area generated by the formatting object must clear every side-float whose parent reference-area is the nearest ancestor reference-area of the generated area, provided the side-float was generated by an fo:float that is before this formatting object, using pre-order traversal order of the formatting tree. Additionally specifies that each area generated by the formatting object must be placed so that the reference-area chain containing the generated area's nearest ancestor reference-area does not contain a later reference-area that is the parent of a side-float generated by an fo:float that is before this formatting object, using pre-order traversal of the formatting tree.

none

This property does not impose any constraints.

When a block-level formatting object is constrained by the "clear" property, its space-before property component values may be altered as necessary for each area that it generates, in order to meet the constraint. The alterations are constrained to produce the minimum additional space required to meet the constraint of the "clear" property.

NOTE: Depending on how near a side-float's after-edge is to the after-edge of its parent reference-area, a block-level formatting object may not be able to generate an area that is a child of the same parent reference-area of the side-float. In this case the first block-area generated by the formatting object must be placed in one of the following columns in the same region or in one of the following pages.

7.16.2. “float”**CSS2 Definition:**

<i>Value:</i>	before start end left right none inherit
<i>Initial:</i>	none
<i>Applies to:</i>	all but positioned elements and generated content
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-float>.

This property specifies whether a box should float to the left, right, or not at all. It may be set for elements that generate boxes that are not absolutely positioned. Values have the following meanings:

left

The element generates a block box that is floated to the left. Content flows on the right side of the box, starting at the top (subject to the "clear" property). The "display" is ignored, unless it has the value "none".

right

Same as "left", but content flows on the left side of the box, starting at the top.

none

The box is not floated.

Here are the precise rules that govern the behavior of floats:

1. The left outer edge of a left-floating box may not be to the left of the left edge of its containing block. An analogous rule holds for right-floating elements.
2. If the current box is left-floating, and there are any left floating boxes generated by elements earlier in the source document, then for each such earlier box, either the left outer edge of the current box must be to the right of the right outer edge of the earlier box, or its top must be lower

than the bottom of the earlier box. Analogous rules hold for right-floating boxes.

3. The right outer edge of a left-floating box may not be to the right of the left outer edge of any right-floating box that is to the right of it. Analogous rules hold for right-floating elements.

4. A floating box's outer top may not be higher than the top of its containing block.

5. The outer top of a floating box may not be higher than the outer top of any block or floated box generated by an element earlier in the source document.

6. The outer top of an element's floating box may not be higher than the top of any line-box containing a box generated by an element earlier in the source document.

7. A left-floating box that has another left-floating box to its left may not have its right outer edge to the right of its containing block's right edge. (Loosely: a left float may not stick out at the right edge, unless it is already as far to the left as possible.) An analogous rule holds for right-floating elements.

8. A floating box must be placed as high as possible.

9. A left-floating box must be put as far to the left as possible, a right-floating box as far to the right as possible. A higher position is preferred over one that is further to the left/right.

XSL modifications to the CSS definition:

The following values have been added for XSL: "before", "start", and "end".

In XSL this property applies only to fo:float

Values have the following meanings:

before

Specifies that the block-areas generated by the fo:float shall be with area-class "xsl-before-float", and shall be descendants of a before-float-reference-area generated by a conditional sub-region of a region-body.

start

Specifies that the block-areas generated by the fo:float shall be with area-class "xsl-side-float" and shall be floated toward the start-edge of the reference area.

end

Specifies that the block-areas returned by the fo:float shall be with area-class "xsl-side-float" and shall be floated toward the end-edge of the reference area.

left

Interpreted as "float='start'".

right

Interpreted as "float='end'".

none

Specifies that the block-areas generated by the fo:float shall be normal.

This property determines the *area-class* trait of the block-areas returned by the fo:float, which controls the placement and parent of these block-areas.

7.17. Keeps and Breaks Properties

Page breaks only apply to descendants of the fo:flow formatting object, and not within absolutely positioned areas, or out-of-line areas. In descendants of fo:flow formatting objects, column breaks apply, and a column break in the last (or only) column implies a page break; column breaks in static-content apply except for those in the last (or only) column which are ignored.

The semantics of keeps and breaks are further described in § 4.8 – [Keeps and Breaks](#) on page 34.

7.17.1. “break-after”

XSL Definition:

<i>Value:</i>	auto column page even-page odd-page inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	block-level formatting objects, fo:list-item, and fo:table-row.
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values for these properties have the following meanings:

auto

No break shall be forced.

NOTE: Page breaks may occur as determined by the formatter's processing as affected by the "widow", "orphan", "keep-with-next", "keep-with-previous", and "keep-together" properties.

column

Imposes a break-after condition with a context consisting of column-areas.

page

Imposes a break-after condition with a context consisting of page-areas.

even-page

Imposes a break-after condition with a context consisting of even page-areas (a blank page may be generated if necessary).

odd-page

Imposes a break-after condition with a context consisting of odd page-areas (a blank page may be generated if necessary).

Specifies that the last area generated by formatting this formatting object shall be the last one placed in a particular context (e.g., page-area, column-area).

This property has no effect when it appears on an fo:table-row formatting object in which there is any

row spanning occurring that includes both the current fo:table-row and the subsequent one.

7.17.2. “break-before”

XSL Definition:

<i>Value:</i>	auto column page even-page odd-page inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	block-level formatting objects, fo:list-item, and fo:table-row.
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

No break shall be forced.

NOTE: Page breaks may occur as determined by the formatter's processing as affected by the "widow", "orphan", "keep-with-next", "keep-with-previous", and "keep-together" properties.

column

Imposes a break-before condition with a context consisting of column-areas.

page

Imposes a break-before condition with a context consisting of page-areas.

even-page

Imposes a break-before condition with a context consisting of even page-areas (a blank page may be generated if necessary).

odd-page

Imposes a break-before condition with a context consisting of odd page-areas (a blank page may be generated if necessary).

Specifies that the first area generated by formatting this formatting object shall be the first one placed in a particular context (e.g., page-area, column-area).

This property has no effect when it appears on an fo:table-row formatting object in which there is any row spanning occurring that includes both the current fo:table-row and the previous one.

7.17.3. “keep-together”

XSL Definition:

<i>Value:</i>	<keep> inherit
<i>Initial:</i>	.within-line=auto, .within-column=auto, .within-page=auto
<i>Applies to:</i>	block-level formatting objects, inline formatting objects, fo:title, fo:list-item,

	fo:list-item-label, and fo:list-item-body
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property imposes keep-together conditions on formatting objects.

The <keep> datatype is composed of three components: within-line, within-column, and within-page. Different components apply to different classes of formatting objects and provide keep conditions relative to different contexts. In the case of the within-line component, the keep context consists of line-areas; for the within-column component, the keep context consists of column-areas; for the within-page component, the keep context consists of page-areas. In the descriptions below, the term "appropriate context" should be interpreted in terms of the previous sentence.

Values of the components have the following meanings:

auto

There are no keep-together conditions imposed by this property.

always

Imposes a keep-together condition with strength "always" in the appropriate context.

<integer>

Imposes a keep-together condition with strength of the given <integer> in the appropriate context.

The semantics of keeps and breaks are further described in § 4.8 – [Keeps and Breaks](#) on page 34.

7.17.4. “keep-with-next”

XSL Definition:

<i>Value:</i>	<keep> inherit
<i>Initial:</i>	.within-line=auto, .within-column=auto, .within-page=auto
<i>Applies to:</i>	block-level formatting objects, inline formatting objects, fo:list-item, and fo:table-row
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property imposes keep-with-next conditions on formatting objects.

The <keep> datatype is composed of three components: within-line, within-column, and within-page. Different components apply to different classes of formatting objects and provide keep conditions relative to different contexts. In the case of the within-line component, the keep context consists of line-areas; for the within-column component, the keep context consists of column-areas; for the within-page component, the keep context consists of page-areas. In the descriptions below, the term "appropriate context" should be interpreted in terms of the previous sentence.

Values of the components have the following meanings:

auto

There are no keep-with-next conditions imposed by this property.

always

Imposes a keep-with-next condition with strength "always" in the appropriate context.

<integer>

Imposes a keep-with-next condition with strength of the given *<integer>* in the appropriate context.

The semantics of keeps and breaks are further described in [§ 4.8 – Keeps and Breaks](#) on page 34.

7.17.5. “keep-with-previous”

XSL Definition:

<i>Value:</i>	<i><keep></i> inherit
<i>Initial:</i>	.within-line=auto, .within-column=auto, .within-page=auto
<i>Applies to:</i>	block-level formatting objects, inline-level formatting objects, fo:list-item, and fo:table-row
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property imposes keep-with-previous conditions on formatting objects.

The *<keep>* datatype is composed of three components: within-line, within-column, and within-page. Different components apply to different classes of formatting objects and provide keep conditions relative to different contexts. In the case of the within-line component, the keep context consists of line-areas; for the within-column component, the keep context consists of column-areas; for the within-page component, the keep context consists of page-areas. In the descriptions below, the term "appropriate context" should be interpreted in terms of the previous sentence.

Values of the components have the following meanings:

auto

There are no keep-with-previous conditions imposed by this property.

always

Imposes a keep-with-previous condition with strength "always" in the appropriate context.

<integer>

Imposes a keep-with-previous condition with strength of the given *<integer>* in the appropriate context.

The semantics of keeps and breaks are further described in [§ 4.8 – Keeps and Breaks](#) on page 34.

7.17.6. “orphans”

CSS2 Definition:

<i>Value:</i>	<integer> inherit
<i>Initial:</i>	2
<i>Applies to:</i>	block-level elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/page.html#propdef-orphans>.

See definition of property widows (§ 7.17.7 – “widows” on page 268).

7.17.7. “widows”

CSS2 Definition:

<i>Value:</i>	<integer> inherit
<i>Initial:</i>	2
<i>Applies to:</i>	block-level elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/page.html#propdef-widows>.

The "orphans" property specifies the minimum number of lines of a paragraph that must be left at the bottom of a page. The "widows" property specifies the minimum number of lines of a paragraph that must be left at the top of a page.

XSL modifications to the CSS definition:

In XSL the "orphans" property specifies the minimum number of line-areas in the first area generated by the formatting object. The "widows" property specifies the minimum number of line-areas in the last area generated by the formatting object.

7.18. Layout-related Properties

The following are layout-related properties that are not common to all formatting objects.

7.18.1. “clip”

CSS2 Definition:

<i>Value:</i>	<shape> auto inherit
<i>Initial:</i>	auto

<i>Applies to:</i>	block-level and replaced elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visufx.html#propdef-clip>.

The 'clip' property applies to elements that have a 'overflow' property with a value other than 'visible'. Values have the following meanings:

auto

The clipping region has the same size and location as the element's box(es).

<shape>

In CSS2, the only valid <shape> value is: rect (<top> <right> <bottom> <left>) where <top>, <bottom> <right>, and <left> specify offsets from the respective sides of the box.

<top>, <right>, <bottom>, and <left> may either have a <length> value or "auto". Negative lengths are permitted. The value "auto" means that a given edge of the clipping region will be the same as the edge of the element's generated box (i.e., "auto" means the same as "0".)

When coordinates are rounded to pixel coordinates, care should be taken that no pixels remain visible when <left> + <right> is equal to the element's width (or <top> + <bottom> equals the element's height), and conversely that no pixels remain hidden when these values are 0.

The element's ancestors may also have clipping regions (in case their "overflow" property is not "visible"); what is rendered is the intersection of the various clipping regions.

If the clipping region exceeds the bounds of the UA's document window, content may be clipped to that window by the native operating environment.

7.18.2. “overflow”

CSS2 Definition:

<i>Value:</i>	visible hidden scroll error-if-overflow auto inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	block-level and replaced elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visufx.html#propdef-overflow>.

This property specifies whether the content of a block-level element is clipped when it overflows the element's box (which is acting as a containing block for the content). Values have the following meanings:

visible

This value indicates that content is not clipped, i.e., it may be rendered outside the block box.

hidden

This value indicates that the content is clipped and that no scrolling mechanism should be provided to view the content outside the clipping region; users will not have access to clipped content. The size and shape of the clipping region is specified by the "clip" property.

scroll

This value indicates that the content is clipped and that if the user agent uses scrolling mechanism that is visible on the screen (such as a scroll bar or a panner), that mechanism should be displayed for a box whether or not any of its content is clipped. This avoids any problem with scrollbars appearing and disappearing in a dynamic environment. When this value is specified and the target medium is "print" or 'projection', overflowing content should be printed.

auto

The behavior of the "auto" value is user agent dependent, but should cause a scrolling mechanism to be provided for overflowing boxes.

Even if "overflow" is set to "visible", content may be clipped to a UA's document window by the native operating environment.

XSL modifications to the CSS definition:

One more value is defined as follows:

error-if-overflow

This value implies the same semantics as the value "hidden" with the additional semantic that an error shall be indicated; implementations may recover by clipping the region.

7.18.3. “reference-orientation”

XSL Definition:

<i>Value:</i>	0 90 180 270 -90 -180 -270 inherit
<i>Initial:</i>	0
<i>Applies to:</i>	see prose
<i>Inherited:</i>	yes (see prose)
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

0

The reference-orientation of this reference-area has the same reference-orientation as the containing reference-area.

90

The reference-orientation of this reference-area is rotated 90 degrees counter-clockwise from the reference-orientation of the containing reference-area.

180

The reference-orientation of this reference-area is rotated 180 degrees counter-clockwise from the reference-orientation of the containing reference-area.

270

The reference-orientation of this reference-area is rotated 270 degrees counter-clockwise from the reference-orientation of the containing reference-area.

-90

The reference-orientation of this reference-area is rotated 270 degrees counter-clockwise from the reference-orientation of the containing reference-area.

NOTE: This is equivalent to specifying "270".

-180

The reference-orientation of this reference-area is rotated 180 degrees counter-clockwise from the reference-orientation of the containing reference-area.

NOTE: This is equivalent to specifying "180".

-270

The reference-orientation of this reference-area is rotated 90 degrees counter-clockwise from the reference-orientation of the containing reference-area.

NOTE: This is equivalent to specifying "90".

The *reference-orientation* specifies the direction for "top" for the content-rectangle of the "reference-area". This is used as the reference for deriving directions, such as the block-progression-direction, inline-progression-direction, etc. as specified by the "writing-mode" and "direction" properties.

The "reference-orientation" property is applied only on formatting objects that establish a reference-area. Each value of "reference-orientation" sets the absolute direction for "top", "left", "bottom", and "right"; which is used by "writing-mode", "direction", and all positioning operations that are referenced to the reference-area or are nested within it.

The *reference-orientation* trait on an area is indirectly derived from the "reference-orientation" property on the formatting object that generates the area or the formatting object ancestors of that formatting object.

7.18.4. “relative-position”

A Property Derived from a CSS2 Property.

<i>Value:</i>	auto static relative inherit
<i>Initial:</i>	static
<i>Applies to:</i>	all block-level and inline-level formatting objects
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

static

The area is normally stacked.

relative

The area's position is determined as if it was normally stacked. Only during rendering is the area rendered offset relative to this position. The fact that one area is relatively positioned does not influence the position on any other area.

For areas that break over a page boundary, only the portion that would have been on a given page originally is included in the repositioned area on that page. Any portion of the repositioned area that was originally on the current page, but falls off the current page due to repositioning is "off" (typically clipped), thus does not fall onto any other page.

7.18.5. “span”

XSL Definition:

<i>Value:</i>	none all inherit
<i>Initial:</i>	none
<i>Applies to:</i>	see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

none

This object does not span multiple columns.

all

The areas resulting from this flow object shall span all the columns of a multi-column region.

Specifies if a block-level object should be placed in the current column or should span all columns of a multi-column region.

7.19. Leader and Rule Properties

7.19.1. “leader-alignment”

XSL Definition:

<i>Value:</i>	none reference-area page inherit
<i>Initial:</i>	none
<i>Applies to:</i>	fo:leader
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

none

Leader-pattern has no special alignment.

reference-area

Leader-pattern is aligned as if it began on the current reference-area's content-rectangle start-edge.

page

Leader-pattern is aligned as if it began on the current page's start-edge.

Specifies whether fo:leaders having identical content and property values shall have their patterns aligned with each other, with respect to their common reference-area or page.

For fo:leaders where the "leader-pattern" property is specified as "dot" or as "use-content", this property will be honored.

If the fo:leader is aligned, the start-edge of each cycle of the repeated pattern will be placed on the start-edge of the next cycle in the appropriate pattern-alignment grid.

7.19.2. “leader-pattern”

XSL Definition:

<i>Value:</i>	space rule dots use-content inherit
<i>Initial:</i>	space
<i>Applies to:</i>	fo:leader
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

space

Leader is to be filled with blank space.

rule

Leader is to be filled with a rule.

If this choice is selected, the "rule-thickness" and "rule-style" properties are used to set the leader's style.

dots

Leader is to be filled with a repeating sequence of dots. The choice of dot character is dependent on the user agent.

use-content

Leader is to be filled with a repeating pattern as specified by the children of the fo:leader.

Provides the specification of how to fill in the leader.

If the leader is aligned, the start-edge of each cycle of each repeating pattern component will be placed on the start-edge of the next cycle in the pattern-alignment grid.

7.19.3. "leader-pattern-width"

XSL Definition:

<i>Value:</i>	use-font-metrics <length> inherit
<i>Initial:</i>	use-font-metrics
<i>Applies to:</i>	fo:leader
<i>Inherited:</i>	yes
<i>Percentages:</i>	Refer to width of containing box
<i>Media:</i>	visual

Values have the following meanings:

use-font-metrics

Use the width of the leader-pattern as determined from its font metrics.

<length>

Sets length for leader-pattern-repeating.

The leader will have an inline-space inserted after each pattern cycle to account for any difference between the width of the pattern as determined by the font metrics and the width specified in this property.

If the length specified is less than the value that would be determined via the use-font-metrics choice, the value of this property is computed as if use-font-metrics choice had been specified.

Specifies the length of each repeat cycle in a repeating leader.

For leaders where the "leader-pattern" property is specified as "dot" or as "use-content", this property will be honored.

7.19.4. “leader-length”

XSL Definition:

<i>Value:</i>	<length-range> inherit
<i>Initial:</i>	leader-length.minimum=0pt, .optimum=12.0pt, .maximum=100%
<i>Applies to:</i>	fo:leader
<i>Inherited:</i>	yes
<i>Percentages:</i>	refer to width of content-rectangle of parent area
<i>Media:</i>	visual

Values have the following meanings:

<length-range>

leader-length.minimum=sets minimum length for a leader

leader-length.optimum=sets optimum length for a leader

leader-length.maximum=sets maximum length for a leader

Specifies the minimum, optimum, and maximum length of an fo:leader.

This property constrains the length of the leader to be between the minimum and maximum lengths.

NOTE: User agents may choose to use the value of "leader-length.optimum" to determine where to break the line, then use the minimum and maximum values during line justification."

7.19.5. “rule-style”

XSL Definition:

<i>Value:</i>	none dotted dashed solid double groove ridge inherit
<i>Initial:</i>	solid
<i>Applies to:</i>	fo:leader
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the style (pattern) of the rule.

This property applies only if the "leader-pattern" property is specified as "rule".

Values have the following meanings:

none

No rule, forces rule-thickness to 0.

dotted

The rule is a series of dots.

dashed

The rule is a series of short line segments.

solid

The rule is a single line segment.

double

The rule is two solid lines. The sum of the two lines and the space between them equals the value of "rule-thickness".

groove

The rule looks as though it were carved into the canvas. (Top/left half of the rule's thickness is the color specified; the other half is white.)

ridge

The opposite of "groove", the rule looks as though it were coming out of the canvas. (Bottom/right half of the rule's thickness is the color specified; the other half is white.)

7.19.6. “rule-thickness”

XSL Definition:

<i>Value:</i>	<length>
<i>Initial:</i>	1.0pt
<i>Applies to:</i>	fo:leader
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the overall thickness of the rule.

This property applies only if the "leader-pattern" property is specified as "rule".

Values have the following meanings:

<length>

The "rule-thickness" is always perpendicular to its length-axis.

The rule is thickened equally above and below the line's alignment position. This can be adjusted through the "baseline-shift" property.

7.20. Properties for Dynamic Effects Formatting Objects**7.20.1. “active-state”**

XSL Definition:

<i>Value:</i>	link visited active hover focus
<i>Initial:</i>	no, a value is required

<i>Applies to:</i>	fo:multi-property-set
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

link

This fo:multi-property-set applies if there is a fo:basic-link descendant of the parent fo:multi-properties and that link has not yet been visited.

visited

This fo:multi-property-set applies if there is a fo:basic-link descendant of the parent fo:multi-properties and that link has been visited.

active

This fo:multi-property-set applies while a normal area returned by the parent fo:multi-properties is being activated by the user. For example, between the times the user presses the mouse button and releases it.

hover

This fo:multi-property-set applies while the user designates a normal area returned by the parent fo:multi-properties (with some pointing device), but does not activate it. For example the cursor (mouse pointer) hovers over such an area.

focus

This fo:multi-property-set applies while a normal area returned by the parent fo:multi-properties has the focus (accepts keyboard events or other forms of text input).

The "active-state" property is used to control which of the fo:multi-property-sets are used to format the child flow objects within an fo:multi-properties formatting object. The states (or at least the events that cause the state to be entered) are defined by the DOM.

7.20.2. "auto-restore"

XSL Definition:

<i>Value:</i>	yes no
<i>Initial:</i>	no
<i>Applies to:</i>	fo:multi-switch
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

yes If this fo:multi-switch is contained in another fo:multi-switch, and that fo:multi-switch changes the active fo:multi-case (hiding this fo:multi-switch), then this fo:multi-switch should restore its initial fo:multi-case.

no This fo:multi-switch should retain its current fo:multi-case.

Specifies if the initial fo:multi-case should be restored when the fo:multi-switch gets hidden by an ancestor fo:multi-switch.

NOTE: A common case of using this property with a "yes" value is when several nested fo:multi-switch objects build an expandable/collapsible table-of-contents view. If the table-of-contents is expanded far down the hierarchy, and an (far above) ancestor is closed, one would want all sub-titles to have restored to their original state when that ancestor is opened again.

7.20.3. “case-name”

XSL Definition:

<i>Value:</i>	<name>
<i>Initial:</i>	none, a value is required
<i>Applies to:</i>	fo:multi-case
<i>Inherited:</i>	no, a value is required
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

<name>

Specifies a name for an fo:multi-case. The name must be unique among the current fo:multi-case siblings, i.e., in the scope of the fo:multi-switch object that (directly) contains them. Other instances of fo:multi-switch objects may use the same names for its fo:multi-case objects.

The purpose of this property is to allow fo:multi-toggle objects to select fo:multi-case objects to switch to.

7.20.4. “case-title”

XSL Definition:

<i>Value:</i>	<string>
<i>Initial:</i>	none, a value is required
<i>Applies to:</i>	fo:multi-case
<i>Inherited:</i>	no, a value is required
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

<string>

Specifies a descriptive title for the fo:multi-case. The title can be displayed in a menu to represent this fo:multi-case when an fo:multi-toggle object names several fo:multi-case objects as allowed destinations.

7.20.5. “destination-placement-offset”

XSL Definition:

<i>Value:</i>	<length>
<i>Initial:</i>	Opt
<i>Applies to:</i>	fo:basic-link
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

<length>

The "destination-placement-offset" property specifies the distance from the beginning (top) of the page to the innermost line-area that contains the first destination area. If the first destination area is not contained in a line-area, the "destination-placement-offset" property instead directly specifies the distance to the top of the destination area.

If the specification of destination-placement-offset would result in a distance longer than the distance from the start of the document, the distance from the start of the document should be used.

If the specified distance would push the first destination area below the page-area, the distance should be decreased so the whole first destination area becomes visible, if possible. If the first destination area is higher than the page, the top of the area should be aligned with the top of the page.

7.20.6. “external-destination”

XSL Definition:

<i>Value:</i>	<uri-specification>
<i>Initial:</i>	null string
<i>Applies to:</i>	fo:basic-link
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

<uri-specification>

Specifies the destination resource (or, when a fragment identifier is given, sub-resource) for an fo:basic-link.

How the destination (sub-)resource is used and/or displayed is application and implementation-dependent. In typical browsing applications, the destination resource is displayed in the browser positioned so that some rendered portion resulting from the processing of some part of the specific destination sub-resource indicated by the fragment identifier is in view.

At least one of the external-destination and internal-destination properties should be assigned. If both are assigned, the system may either report the error, or use the internal-destination property.

7.20.7. “indicate-destination”

XSL Definition:

<i>Value:</i>	yes no
<i>Initial:</i>	no
<i>Applies to:</i>	fo:basic-link
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

yes The areas that belong to the link target when traversed should, in a system-dependent manner, be indicated.

no No special indication should be made.

NOTE: This could be indicated in any feasible way, e.g., by reversed video, etc.

7.20.8. “internal-destination”

XSL Definition:

<i>Value:</i>	null string <idref>
<i>Initial:</i>	null string
<i>Applies to:</i>	fo:basic-link
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

<idref>

Specifies the destination flow object of an fo:basic-link. This property allows the destination flow object node to be explicitly specified.

At least one of the external-destination and internal-destination properties should be assigned. If both are assigned, the system may either report the error, or use the internal-destination property.

7.20.9. “show-destination”

XSL Definition:

<i>Value:</i>	replace new
<i>Initial:</i>	replace
<i>Applies to:</i>	fo:basic-link
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A

Media: interactive

replace

The current document view should be replaced. However, if the destination area(s) are already available in a page/region, those areas should simply be moved/scrolled "into sight".

new

A new (additional) document view should always be opened.

Specifies where the destination resource should be displayed.

7.20.10. “starting-state”

XSL Definition:

Value: show | hide
Initial: show
Applies to: fo:multi-case
Inherited: no
Percentages: N/A
Media: interactive

show

The content of the fo:multi-case is a candidate for being displayed initially.

hide

The content of the fo:multi-case is not a candidate for being displayed initially.

Specifies if the fo:multi-case can be initially displayed.

The parent fo:multi-switch shall choose the first fo:multi-case child where the property "starting-state" has the value equal to "show".

NOTE: Any number of the fo:multi-case objects may assign "starting-state" to "show".

If no fo:multi-case has "starting-state" property value of "show", the contents of no fo:multi-case should be displayed.

NOTE: If no multi-case is displayed, the entire fo:multi-switch will effectively be hidden.

7.20.11. “switch-to”

XSL Definition:

Value: xsl-preceding | xsl-following | xsl-any | <name>[<name>]*
Initial: xsl-any
Applies to: fo:multi-toggle

<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

xsl-preceding

Activating the switch should result in the current fo:multi-case being replaced by its preceding sibling.

NOTE: The current fo:multi-case is the closest ancestor fo:multi-case.

In other words, the current fo:multi-switch should switch to the previous sibling of the fo:multi-case that is currently selected.

NOTE: The current fo:multi-switch is the closest ancestor fo:multi-switch.

If the current fo:multi-case is the first sibling, xsl-preceding should switch to the last fo:multi-case sibling.

xsl-following

Activating the switch should result in that the current fo:multi-case is replaced by its next sibling.

If the current fo:multi-case is the last sibling, xsl-following should switch to the first fo:multi-case sibling.

xsl-any

Activating the switch should allow the user to select any other fo:multi-case sibling.

If there is only a single other fo:multi-case, the toggle should immediately switch to it (and not show that single choice to the user).

<name>

A name matching a case-name of an fo:multi-case.

Specifies what fo:multi-case object(s) this fo:multi-toggle shall switch to.

If switch-to is a name list, the user can switch to any of the named multi-case objects. If a multi-toggle with a single name is activated, it should immediately switch to the named multi-case.

NOTE: How to actually select the multi-case from a list is system dependent.

7.20.12. “target-presentation-context”

XSL Definition:

<i>Value:</i>	use-target-processing-context <uri-specification>
<i>Initial:</i>	use-target-processing-context
<i>Applies to:</i>	fo:basic-link
<i>Inherited:</i>	no

Percentages: N/A
Media: interactive

use-target-processing-context

The context specified by the "target-processing-context" property shall be used.

<uri-specification>

Specifies the limited context in which the resource should be presented if the external destination is a resource of a processed structured media type for which a limited presentational context makes sense (e.g., XML, XHTML, SVG).

This property is ignored if the "external-destination" property has a null string value or if the external destination is not of a processed structured media type for which a limited presentational context makes sense.

NOTE: For example, an XML and XSL implementation may parse the XML document, but begin XSLT processing by applying templates to the node set indicated by the "target-presentation-context" property.

NOTE: If this is a node other than the document root, numbering and other contextually-dependent presentation may differ between implementations. Some implementations may want to make this tradeoff for memory or performance reasons.

7.20.13. “target-processing-context”

XSL Definition:

Value: document-root | <uri-specification>
Initial: document-root
Applies to: fo:basic-link
Inherited: no
Percentages: N/A
Media: interactive

document-root

The root of the document of the external-destination is used.

<uri-specification>

Specifies the root of a virtual document that the processor preparing the new presentation should process if the external destination is a resource of a processed structured media type (e.g., XML, SVG).

This property is ignored if the "external-destination" property has a null string value or if the external destination is not of a processed structured media type.

NOTE: Not all URI references will be sensible roots, e.g., an XPointer that gives a string range into an XML document.

If the root is not valid for the media type the processor may ignore this property.

7.20.14. “target-stylesheet”

XSL Definition:

<i>Value:</i>	use-normal-stylesheet <uri-specification>
<i>Initial:</i>	use-normal-stylesheet
<i>Applies to:</i>	fo:basic-link
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	interactive

use-normal-stylesheet

The implementation will discover stylesheets using its usual methods.

<uri-specification>

Specifies the stylesheet that shall be used for processing the resource. This stylesheet shall be used instead of any other stylesheet that otherwise would be used.

NOTE: For example from HTTP header information, XML stylesheet processing instructions, or XHTML style and link elements.

This property is ignored if the "external-destination" property has a null string value or if the external destination is not of a media type that uses stylesheets.

NOTE: In this version of XSL, only a single stylesheet URI reference is permitted. A future version of XSL may extend the stylesheet specification.

7.21. Properties for Markers

7.21.1. “marker-class-name”

XSL Definition:

<i>Value:</i>	<name>
<i>Initial:</i>	an empty name
<i>Applies to:</i>	fo:marker
<i>Inherited:</i>	no, a value is required
<i>Percentages:</i>	N/A
<i>Media:</i>	paged

Values have the following meanings:

<name>

Names used as identifiers must be unique among fo:markers that are (conceptually) attached to the same area.

If the name is empty or if a name-conflict is encountered, an error shall be reported. A processor may then continue processing.

This property identifies the fo:marker as being in a group with others that have the same name, each of which becomes a candidate to be retrieved by an fo:retrieve-marker that has a "retrieve-class-name" property of the same value.

7.21.2. “retrieve-class-name”

XSL Definition:

<i>Value:</i>	<name>
<i>Initial:</i>	an empty name
<i>Applies to:</i>	fo:retrieve-marker
<i>Inherited:</i>	no, a value is required
<i>Percentages:</i>	N/A
<i>Media:</i>	paged

Values have the following meanings:

<name>

A name that matches the "marker-class-name" property value of an fo:marker.

This property constrains that the fo:marker whose children are retrieved by the fo:retrieve-marker must have a "marker-class-name" property value that is the same as the value of this property.

7.21.3. “retrieve-position”

XSL Definition:

<i>Value:</i>	first-starting-within-page first-including-carryover last-starting-within-page last-ending-within-page
<i>Initial:</i>	first-starting-within-page
<i>Applies to:</i>	fo:retrieve-marker
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	paged

The term "containing page" is used here to mean the page that contains the first area generated or returned by the children of the retrieved fo:marker.

Values have the following meanings:

first-starting-within-page

Specifies a preference for retrieving the children of an fo:marker attached to an area that is

- within the containing page
- whose "is-first" trait is set to "true"

and that precedes in the area tree any other similarly constrained area that is attached to an identically named fo:marker, using pre-order traversal order.

first-including-carryover

Specifies a preference for retrieving the children of an fo:marker attached to an area that is within the containing page and that precedes in the area tree any other similarly constrained area that is attached to an identically named fo:marker, using pre-order traversal order.

last-starting-within-page

Specifies a preference for retrieving the children of an fo:marker attached to an area that is

- within the containing page
- whose "is-first" trait is set to "true"

and that follows in the area tree any other similarly constrained area that is attached to an identically named fo:marker, using pre-order traversal order.

last-ending-within-page

Specifies a preference for retrieving the children of an fo:marker attached to an area that is within the containing page whose "is-last" trait is set to "true" and that follows in the area tree any other similarly constrained area that is attached to an identically named fo:marker, using pre-order traversal order.

This property specifies the preference for which fo:marker's children shall be retrieved by an fo:retrieve-marker, based on the areas returned by the parent of the fo:marker relative to the areas returned by the parents of other identically named fo:marker's.

7.21.4. “retrieve-boundary”

XSL Definition:

<i>Value:</i>	page page-sequence document
<i>Initial:</i>	page-sequence
<i>Applies to:</i>	fo:retrieve-marker
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	paged

The term "containing page" is used here to mean the page that contains the first area generated or returned by the children of the retrieved fo:marker.

Values have the following meanings:

page

Specifies that the children of any fo:markers whose parent generated or returned a normal area within the containing page or generated non-normal area within the containing page may be retrieved by this fo:retrieve-marker.

page-sequence

Specifies that only the children of fo:markers that are descendants of the fo:flow within the containing fo:page-sequence may be retrieved by this fo:retrieve-marker.

document

Specifies that the children of any fo:marker that is a descendant of any fo:flow within the document may be retrieved by this fo:retrieve-marker.

7.22. Properties for Number to String Conversion

7.22.1. “format”

XSL Definition:

<i>Value:</i>	<string>
<i>Initial:</i>	1.
<i>Applies to:</i>	fo:page-sequence
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	all

This property is defined in [XSLT]: Number to String Conversion Attributes.

7.22.2. “grouping-separator”

XSL Definition:

<i>Value:</i>	<character>
<i>Initial:</i>	no separator
<i>Applies to:</i>	fo:page-sequence
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	all

This property is defined in [XSLT]: Number to String Conversion Attributes.

7.22.3. “grouping-size”

XSL Definition:

<i>Value:</i>	<number>
<i>Initial:</i>	no grouping
<i>Applies to:</i>	fo:page-sequence
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	all

This property is defined in [XSLT]: Number to String Conversion Attributes.

7.22.4. “letter-value”

XSL Definition:

<i>Value:</i>	auto alphabetic traditional
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:page-sequence
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	all

This property is defined in [XSLT]: Number to String Conversion Attributes. A value of "auto" corresponds to the XSLT definition for when the attribute is not specified.

7.23. Pagination and Layout Properties

The following pagination and layout properties are all XSL only.

7.23.1. “blank-or-not-blank”

XSL Definition:

<i>Value:</i>	blank not-blank any inherit
<i>Initial:</i>	any
<i>Applies to:</i>	fo:conditional-page-master-reference
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property forms part of a selection rule to determine if the referenced page-master is eligible for selection at this point in the page-sequence.

The values have the following meanings:

blank

This master is eligible for selection if a page must be generated (e.g., to maintain proper page parity at the start or end of the page-sequence) and there are no areas from the fo:flow to be put on that page.

not-blank

This master is eligible for selection if this page contains areas from the fo:flow.

any

This master is always eligible for selection.

7.23.2. “column-count”

XSL Definition:

<i>Value:</i>	<number> inherit
<i>Initial:</i>	1
<i>Applies to:</i>	fo:region-body
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<integer>

A positive integer. If a negative or non-integer value is provided, the value will be rounded to the nearest integer value greater than or equal to 1.

Specifies the number of columns in the region.

A value of 1 indicates that this is not a multi-column region.

7.23.3. “column-gap”

XSL Definition:

<i>Value:</i>	<length> <percentage> inherit
<i>Initial:</i>	12.0pt
<i>Applies to:</i>	fo:region-body
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of the region being divided into columns.
<i>Media:</i>	visual

Values have the following meanings:

<length>

This is an unsigned length. If a negative value has been specified a value of 0pt will be used.

<percentage>

The value is a percentage of the inline-progression-dimension of the content-rectangle of the region.

Specifies the width of the separation between adjacent columns in a multi-column region. See the description in § 6.4.13 – [fo:region-body](#) on page 86 for further details.

7.23.4. “extent”

XSL Definition:

<i>Value:</i>	<length> <percentage> inherit
---------------	-----------------------------------

<i>Initial:</i>	0.0pt
<i>Applies to:</i>	fo:region-before, fo:region-after, fo:region-start, fo:region-end
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to the corresponding height or width of the region reference-area.
<i>Media:</i>	visual

Values have the following meanings:

<length>

This is an unsigned length. If a negative value has been specified a value of 0pt will be used.

<percentage>

The value is a percentage of corresponding height or width of the page.

Specifies the width of the region-start or region-end or the height of the region-before or region-after.

7.23.5. “flow-name”

XSL Definition:

<i>Value:</i>	<name>
<i>Initial:</i>	an empty name
<i>Applies to:</i>	fo:flow, fo:static-content
<i>Inherited:</i>	no, a value is required
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<name>

Names used as identifiers must be unique within an fo:page-sequence.

If the name is empty or if a name-conflict is encountered, an error shall be reported. A processor may then continue processing.

Defines the name of the flow.

The flow-name and region-name are used to assign the flow's content (or static-content's content) to a specific region or series of regions in the layout. In XSL this is done by specifying the name of the target region as the flow-name. (For example, text placed in the region-body would specify flow-name="xsl-region-body".)

7.23.6. “force-page-count”

XSL Definition:

<i>Value:</i>	auto even odd end-on-even end-on-odd no-force inherit
<i>Initial:</i>	auto

<i>Applies to:</i>	fo:page-sequence
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Force-page-count is used to impose a constraint on the number of pages in a page-sequence. In the event that this constraint is not satisfied, an additional page will be added to the end of the sequence. This page becomes the "last" page of that sequence.

The values have the following meanings:

auto

Force the last page in this page-sequence to be an odd-page if the initial-page-number of the next page-sequence is even. Force it to be an even-page if the initial-page-number of the next page-sequence is odd. If there is no next page-sequence or if the value of its initial-page-number is "auto" do not force any page.

even

Force an even number of pages in this page-sequence.

odd

Force an odd number of pages in this page-sequence.

end-on-even

Force the last page in this page-sequence to be an even-page.

end-on-odd

Force the last page in this page-sequence to be an odd-page.

no-force

Do not force either an even or an odd number of pages in this page-sequence

NOTE: Whether a page is an odd-page or even-page is determined from the *page-number* trait.

7.23.7. “initial-page-number”

XSL Definition:

<i>Value:</i>	auto auto-odd auto-even <number> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:page-sequence
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

The initial number shall be set to 1 if no previous fo:page-sequence exists in the document.

If a preceding page-sequence exists, the initial number will be one greater than the last number for that sequence.

auto-odd

A value is determined in the same manner as for "auto". If that value is an even number 1 is added.

auto-even

A value is determined in the same manner as for "auto". If that value is an odd number 1 is added.

<integer>

A positive integer. If a negative or non-integer value is provided, the value will be rounded to the nearest integer value greater than or equal to 1.

Sets the initial-page-number to be used on this page-sequence.

7.23.8. “master-name”

XSL Definition:

<i>Value:</i>	<name>
<i>Initial:</i>	an empty name
<i>Applies to:</i>	fo:page-sequence, fo:simple-page-master, fo:page-sequence-master, fo:single-page-master-reference, fo:repeatable-page-master-reference, fo:conditional-page-master-reference
<i>Inherited:</i>	no, a value is required
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<name>

Names used as master identifiers may not be empty and must be unique.

Uses of the name to reference a given named object need not be unique, but may not be empty and must refer to a master-name that exists within the document.

This property is used for two purposes:

- Identifying a master:

If this property is specified on an fo:simple-page-master, it provides an identifying name of the master. This name is subsequently referenced as the value of properties on the following formatting objects: fo:single-page-master-reference, fo:repeatable-page-master-reference, and

fo:conditional-page-master-reference to request the use of this master when creating a page instance. It may also be used on an fo:page-sequence to specify the use of this master when creating page instances.

If this property is specified on an fo:page-sequence-master, it provides an identifying name of the master. This name is subsequently referenced as the value of properties on the fo:page-sequence to request the use of this page-sequence-master when creating page instances.

A master-name must be unique across all page-masters and page-sequence-masters.

If the name is empty or if a name-conflict is encountered, an error shall be reported. A processor may then continue processing.

- Selecting a master:
 - If this property is specified on the fo:page-sequence it specifies the name of the page-sequence-master or page-master to be used to create pages in the sequence.
 - If this property is specified on the fo:single-page-master-reference, it specifies the name of the page-master to be used to create a single page instance.
 - If this property is specified on the fo:repeatable-page-master-reference, it specifies the name of the page-master to be used in repetition until the content is exhausted or the maximum-repeats limit is reached, whichever occurs first.
 - If this property is specified on the fo:conditional-page-master-reference, it specifies the name of the page-master to be used whenever this alternative is chosen.

If the name is empty or if a name-conflict is encountered, an error shall be reported. A processor may then continue processing.

7.23.9. “maximum-repeats”

XSL Definition:

<i>Value:</i>	<number> no-limit inherit
<i>Initial:</i>	no-limit
<i>Applies to:</i>	fo:repeatable-page-master-reference, fo:repeatable-page-master-alternatives
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the constraint on the maximum number of pages in the sub-sequence of pages that may be generated by an fo:page-sequence that uses the fo:repeatable-page-master-reference or fo:repeatable-page-master-alternatives on which this property is specified.

The values have the following meanings:

no-limit

No constraint is specified.

<integer>

The maximum number of pages in the sub-sequence.

The value is an integer greater than or equal to 0.

If a fractional value or a value less than 0 is specified, it will be rounded to the nearest integer greater than or equal to 0.

A value of 0 indicates this master-reference will not be used.

7.23.10. “odd-or-even”

XSL Definition:

<i>Value:</i>	odd even any inherit
<i>Initial:</i>	any
<i>Applies to:</i>	fo:conditional-page-master-reference
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property forms part of a selection rule to determine if the referenced page-master is eligible for selection at this point in the page-sequence.

The values have the following meanings:

odd

This master is eligible for selection if the page number is odd.

even

This master is eligible for selection if the page number is even.

any

This master is eligible for selection regardless of whether the page number is odd or even.

NOTE: "Page number" refers to the *page-number* trait for the page to be generated.

7.23.11. “page-height”

XSL Definition:

<i>Value:</i>	auto indefinite <length> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:simple-page-master
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

The "page-height" shall be determined by the formatter from the height of the media or browser window. If media information is not available this dimension shall be implementation-defined.

NOTE: A fallback to 11.0in would fit on both 8+1/2x11 and A4 pages).

indefinite

The height of the page is determined from the size of the laid-out content.

"Page-width" and "page-height" may not both be set to "indefinite". Should that occur, the dimension that is parallel to the block-progression-direction, as determined by the "reference-orientation" and "writing-mode" on the fo:simple-page-master, of the page-reference-area will remain "indefinite" and the other will revert to "auto".

<length>

Specifies a fixed height for the page.

Specifies the height of a page.

7.23.12. “page-position”

XSL Definition:

<i>Value:</i>	first last rest any inherit
<i>Initial:</i>	any
<i>Applies to:</i>	fo:conditional-page-master-reference
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property forms part of a selection rule to determine if the referenced page-master is eligible for selection at this point in the page-sequence.

The values have the following meanings:

first

This master is eligible for selection if this is the first page in the page-sequence.

last

This master is eligible for selection if this is the last page in the page-sequence.

rest

This master is eligible for selection if this is not the first page nor the last page in the page-sequence.

any

This master is eligible for selection regardless of page positioning within the page-sequence.

7.23.13. “page-width”

XSL Definition:

<i>Value:</i>	auto indefinite <length> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:simple-page-master
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

The "page-width" shall be determined by the formatter from the width of the media or browser window. If media information is not available this dimension shall be implementation-defined.

NOTE: A fallback to 8.26in would fit on both 8+1/2x11 and A4 pages).

indefinite

The width of the page is determined from the size of the laid-out content.

"Page-width" and "page-height" properties may not both be set to "indefinite". Should that occur, the dimension that is parallel to the block-progression-direction, as determined by the "reference-orientation" and "writing-mode" on the fo:simple-page-master, of the page-reference-area will remain "indefinite" and the other will revert to "auto".

<length>

Specifies a fixed width for the page.

Specifies the width of a page.

7.23.14. “precedence”

XSL Definition:

<i>Value:</i>	true false inherit
<i>Initial:</i>	false
<i>Applies to:</i>	fo:before-region, fo:after-region, fo:start-region, fo:end-region
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

false

A value of false specifies that this region does not extend to the page margins. This region has the same width or height as the body.

true

A value of true specifies that this region takes precedence and extends across the full size of the page or view.

Specifies which region (i.e., region-before, region-after, region-start, or region-end) takes precedence in terms of which may extend into the corners of the simple-page-master.

If both adjacent regions have equal precedence, the before-region or after-region is treated as if 'true' had been specified and the start-region or end-region will be treated as if 'false' had been specified.

7.23.15. “region-name”

XSL Definition:

<i>Value:</i>	xsl-region-body xsl-region-start xsl-region-end xsl-region-before xsl-region-after xsl-before-float-separator xsl-footnote-separator <name>
<i>Initial:</i>	see prose
<i>Applies to:</i>	fo:region-body, fo:region-start, fo:region-end, fo:region-before, and fo:region-after
<i>Inherited:</i>	no, a value is required
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

xsl-region-body

Reserved region-name for use as default name of fo:region-body. This name may not be used on any other class of region.

xsl-region-start

Reserved region-name for use as default name of fo:region-start. This name may not be used on any other class of region.

xsl-region-end

Reserved region-name for use as default name of fo:region-end. This name may not be used on any other class of region.

xsl-region-before

Reserved region-name for use as default name of fo:region-before. This name may not be used on any other class of region.

xsl-region-after

Reserved region-name for use as default name of fo:region-after. This name may not be used on any other class of region.

xsl-before-float-separator

Reserved name for content to be used as a separator at the end-edge of a before-float-reference-area. This name may not be used on any other class of region.

xsl-footnote-separator

Reserved name for content to be used as a separator at the start-edge of a footnote-reference-area. This name may not be used on any other class of region.

<name>

Names used as identifiers must be unique within a page-master.

This property is used to identify a region within a simple-page-master.

The "region-name" may be used to differentiate a region that lies on a page-master for an odd page from a region that lies on a page-master for an even page. In this usage, once a name is used for a specific class of region (start, end, before, after, or body), that name may only be used for regions of the same class in any other page-master. The reserved names may only be used in the manner described above.

7.24. Table Properties

7.24.1. "border-after-precedence"

XSL Definition:

<i>Value:</i>	force <integer> inherit
<i>Initial:</i>	fo:table: 1, fo:table-cell: 0
<i>Applies to:</i>	fo:table, fo:table-cell
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

force

The precedence is higher than any <integer>.

<integer>

A numeric precedence specification. A higher value has a higher precedence than a lower one.

Specifies the precedence of the border specification on this formatting object for the border-after.

7.24.2. "border-before-precedence"

XSL Definition:

<i>Value:</i>	force <integer> inherit
---------------	-----------------------------

<i>Initial:</i>	fo:table: 1, fo:table-cell: 0
<i>Applies to:</i>	fo:table, fo:table-cell
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the precedence of the border specification on this formatting object for the border-before.

See definition of property border-after-precedence (§ 7.24.1 – “border-after-precedence” on page 298).

7.24.3. “border-collapse”

CSS2 Definition:

<i>Value:</i>	collapse separate inherit
<i>Initial:</i>	collapse
<i>Applies to:</i>	table
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/tables.html#propdef-border-collapse>.

collapse

The value "collapse" selects the collapsing borders model.

separate

The value "separate" selects the separated borders border model.

This property selects a table's border model. The value "separate" selects the separated borders border model. The value "collapse" selects the collapsing borders model.

7.24.4. “border-end-precedence”

XSL Definition:

<i>Value:</i>	force <integer> inherit
<i>Initial:</i>	fo:table: 1, fo:table-cell: 0
<i>Applies to:</i>	fo:table, fo:table-cell
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the precedence of the border specification on this formatting object for the border-end.

See definition of property border-after-precedence (§ 7.24.1 – “border-after-precedence” on page 298).

7.24.5. “border-separation”

XSL Definition:

<i>Value:</i>	<length-bp-ip-direction> inherit
<i>Initial:</i>	.block-progression-direction="0pt" .inline-progression-direction="0pt"
<i>Applies to:</i>	table
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

<length-bp-ip-direction>

The lengths specify the distance that separates adjacent cell borders in the row-stacking-direction (given by the block-progression-direction of the table), and in the column-stacking-direction (given by the inline-progression-direction of the table).

In the separate borders model, each cell has an individual border. The "border-separation" property specifies the distance between the borders of adjacent cells. This space is filled with the background of the table element. Rows, columns, row groups, and column groups cannot have borders (i.e., user agents must ignore the border properties for those elements).

7.24.6. “border-start-precedence”

XSL Definition:

<i>Value:</i>	force <integer> inherit
<i>Initial:</i>	fo:table: 1, fo:table-cell: 0
<i>Applies to:</i>	fo:table, fo:table-cell
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Specifies the precedence of the border specification on this formatting object for the border-start.

See definition of property border-after-precedence (§ 7.24.1 – “border-after-precedence” on page 298).

7.24.7. “caption-side”

CSS2 Definition:

<i>Value:</i>	before after start end top bottom left right inherit
<i>Initial:</i>	before
<i>Applies to:</i>	fo:table-caption
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/tables.html#q6>.

top

Positions the caption box above the table box.

bottom

Positions the caption box below the table box.

left

Positions the caption box to the left of the table box.

right

Positions the caption box to the right of the table box.

This property specifies the position of the caption box with respect to the table box.

Captions above or below a "table" element are formatted very much as if they were a block element before or after the table, except that (1) they inherit inheritable properties from the table, and (2) they are not considered to be a block box for the purposes of any "compact" or "run-in" element that may precede the table.

A caption that is above or below a table box also behaves like a block box for width calculations; the width is computed with respect to the width of the table box's containing block.

For a caption that is on the left or right side of a table box, on the other hand, a value other than "auto" for "width" sets the width explicitly, but "auto" tells the user agent to choose a "reasonable width". This may vary between "the narrowest possible box" to "a single line", so we recommend that users do not specify "auto" for left and right caption widths.

To align caption content horizontally within the caption box, use the "text-align" property. For vertical alignment of a left or right caption box with respect to the table box, use the "vertical-align" property. The only meaningful values in this case are "top", "middle", and "bottom". All other values are treated the same as "top".

XSL modifications to the CSS definition:

Insert the following writing-mode relative values:

before

Positions the caption before the table in the block-progression-direction.

after

Positions the caption after the table in the block-progression-direction.

start

Positions the caption before the table in the inline-progression-direction.

end

Positions the caption after the table in the inline-progression-direction.

The CSS qualifications (1) and (2) do not apply. The last three sentences in the last paragraph (referencing "vertical-align") do not apply.

7.24.8. “column-number”

XSL Definition:

<i>Value:</i>	<number>
<i>Initial:</i>	see prose
<i>Applies to:</i>	fo:table-column, fo:table-cell
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

<integer>

A positive integer. If a negative or non-integer value is provided, the value will be rounded to the nearest integer value greater than or equal to 1.

For an fo:table-column formatting object, it specifies the column-number of the table cells that may use properties from this fo:table-column formatting object by using the from-table-column() function. The initial value is 1 plus the column-number of the previous table-column, if there is a previous table-column, and otherwise 1.

For an fo:table-cell it specifies the number of the first column to be spanned by the table-cell. The initial value is the current column-number. For the first table-cell in a table-row, the current column number is 1. For other table-cells, the current column-number is the column-number of the previous table-cell in the row plus the number of columns spanned by that previous cell.

7.24.9. “column-width”

XSL Definition:

<i>Value:</i>	<length> <percentage>
<i>Initial:</i>	see prose
<i>Applies to:</i>	fo:table-column
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of table
<i>Media:</i>	visual

<length>

The "column-width" property specifies the width of the column whose value is given by the "column-number" property. This property, if present, is ignored if the "number-columns-spanned" property is greater than 1. The "column-width" property must be specified for every column, unless the automatic table layout is used.

NOTE: The result of using a percentage for the width may be unpredictable, especially when using the automatic table layout.

7.24.10. “empty-cells”

CSS2 Definition:

Value: show | hide | inherit

Initial: show

Applies to: table-cell

Inherited: yes

Percentages: N/A

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/tables.html#propdef-empty-cells>.

show

When this property has the value "show", borders are drawn around empty cells (like normal cells).

hide

A value of "hide" means that no borders are drawn around empty cells. Furthermore, if all the cells in a row have a value of "hide" and have no visible content, the entire row behaves as if it had "display: none".

In the separated borders model, this property controls the rendering of borders around cells that have no visible content. Empty cells and cells with the "visibility" property set to "hidden" are considered to have no visible content. Visible content includes " " (non-breaking-space) and other whitespace except ASCII CR ("\0D"), LF ("\0A"), tab ("\09"), and space ("\20").

7.24.11. “ends-row”

XSL Definition:

Value: yes | no

Initial: no

Applies to: fo:table-cell

Inherited: no

Percentages: N/A

Media: visual

yes This cell ends a row.

no This cell does not end a row.

Specifies whether this cell ends a row. This is only allowed for table-cells that are not in table-rows.

7.24.12. “number-columns-repeated”

XSL Definition:

<i>Value:</i>	<number>
<i>Initial:</i>	1
<i>Applies to:</i>	fo:table-column
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

<integer>

A positive integer. If a negative or non-integer value is provided, the value will be rounded to the nearest integer value greater than or equal to 1.

The "number-columns-repeated" property specifies the repetition of a table-column specification *n* times; with the same effect as if the fo:table-column formatting object had been repeated *n* times in the result tree. The "column-number" property, for all but the first, is the column-number of the previous one plus its value of the "number-columns-spanned" property.

NOTE: This handles HTML's "colgroup" element.

7.24.13. “number-columns-spanned”

XSL Definition:

<i>Value:</i>	<number>
<i>Initial:</i>	1
<i>Applies to:</i>	fo:table-column, fo:table-cell
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

<integer>

A positive integer. If a negative or non-integer value is provided, the value will be rounded to the nearest integer value greater than or equal to 1.

For an fo:table-column the "number-columns-spanned" property specifies the number of columns spanned by table-cells that may use properties from this fo:table-column formatting object using the from-table-column() function.

For an fo:table-cell the "number-columns-spanned" property specifies the number of columns which this cell spans in the column-progression-direction starting with the current column.

7.24.14. “number-rows-spanned”

XSL Definition:

<i>Value:</i>	<number>
<i>Initial:</i>	1
<i>Applies to:</i>	fo:table-cell
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

<integer>

A positive integer. If a negative or non-integer value is provided, the value will be rounded to the nearest integer value greater than or equal to 1.

The "number-rows-spanned" property specifies the number of rows which this cell spans in the row-progression-direction starting with the current row.

7.24.15. “starts-row”

XSL Definition:

<i>Value:</i>	yes no
<i>Initial:</i>	no
<i>Applies to:</i>	fo:table-cell
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

yes This cell starts a row.

no This cell does not start a row.

Specifies whether this cell starts a row. This is only allowed for table-cells that are not in table-rows.

NOTE: The "starts-row" and "ends-row" properties with a "yes" value are typically used when the input data does not have elements containing the cells in each row, but instead, for example, each row starts at elements of a particular type.

7.24.16. “table-layout”

CSS2 Definition:

<i>Value:</i>	auto fixed inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	table

Inherited: no
Percentages: N/A
Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/tables.html#propdef-table-layout>.

fixed

Use the fixed table layout algorithm

auto

Use any automatic table layout algorithm

The "table-layout" property controls the algorithm used to lay out the table cells, rows, and columns.

7.24.17. “table-omit-footer-at-break”

XSL Definition:

Value: yes | no
Initial: no
Applies to: fo:table
Inherited: no
Percentages: N/A
Media: visual

yes This property specifies that the footer should be omitted.

no This property specifies that the footer should not be omitted.

The "table-omit-footer-at-break" property specifies if a table whose last area is not at the end of an area produced by the table should end with the content of the fo:table-header formatting object or not.

7.24.18. “table-omit-header-at-break”

XSL Definition:

Value: yes | no
Initial: no
Applies to: fo:table
Inherited: no
Percentages: N/A
Media: visual

yes This property specifies that the header should be omitted.

no This property specifies that the header should not be omitted.

The "table-omit-header-at-break" property specifies if a table whose first area is not at the beginning of an area produced by the table should start with the content of the fo:table-header formatting object or not.

7.25. Writing-mode-related Properties

The properties in this section control the setting of the inline-progression-direction, the block-progression-direction and the orientation of the glyphs that are placed on a baseline in the inline-progression-direction. The "writing-mode" property sets both the "inline-progression-direction" and the "block-progression-direction"s.

The glyph orientation properties, "glyph-orientation-horizontal" and "glyph-orientation-vertical" set the orientation of the glyph relative to the default glyph orientation. The default orientation for glyphs is with the top of the glyph oriented toward the top of the reference area of which the glyph area is a descendant; that is, the glyph orientation is the same as the reference-orientation of the reference area. Glyphs that are oriented at '90' or '-90' degrees from the reference-orientation are said to be *rotated glyphs*. Glyphs that are oriented 180 degrees from the reference-orientation are said to be *inverted glyphs*.

The "direction" property (which is controlled by the "unicode-bidi" property) only affects text in which the orientation of the glyphs is perpendicular to the dominant-baseline. For horizontal writing-modes, the "direction" property only affects character sequences that generate glyphs that are not rotated. For vertical writing-modes, the "direction" property only affects character sequences that generate glyphs that are rotated.

The following sample fragment of XML is used to illustrate the interaction between the "writing-mode," "direction" and "glyph-orientation-vertical" properties.

```
<t> 𐤀
  <t-s1>Apex</t-s1>
  <t-s2>אבגד</t-s2> 美
</t>
```

Markup of text in the next figure

In the XML markup of the figure above, the characters are represented by their presentation form (and not their Unicode character codes). The order in which the characters are depicted is their storage order. The Hebrew characters in the third line are (from left to right) the first four letters of the Hebrew alphabet: aleph, beth, gimel and dalet. The generic identifiers in the XML markup are arbitrary, but are intended to suggest a sequence of text with two embedded text spans.

The following figure shows the effect of specifying an assortment of values for the "direction" and "glyph-orientation-vertical" properties that are specified on the three elements in the above XML fragment. In all cases, the "writing-mode" is "tb-rl". And in all cases the Unicode Bidi Algorithm [UNICODE TR9] is applied to the characters that are the children or descendants of the <t> element, sometimes with explicit directional markup in terms of the "direction" property and other times using the intrinsic direction properties of the underlying characters. The Unicode Bidi Algorithm is applied as the last step of refinement (see § 5 – Property Refinement / Resolution on page 39) and before mapping the characters to glyphs and applying any rotation due to a glyph-orientation value.

The figure shows seven possible presentations of the sample XML fragment, one with all glyphs having a vertical orientation and six with various combinations of a perpendicular glyph-orientation and

Elements/ Cases	<t>	<t-s1>	<t-s2>
(5)	writing-mode: tb-rl glyph-orientation-vertical: 0 direction: rtl	glyph-orientation-vertical: -90 unicode-bidi: bidi-override	glyph-orientation-vertical: 90
(6)	writing-mode: tb-rl glyph-orientation-vertical: 0 direction: rtl	glyph-orientation-vertical: -90 unicode-bidi: bidi-override	glyph-orientation-vertical: -90 unicode-bidi: bidi-override direction: ltr
(7)	writing-mode: tb-rl glyph-orientation-vertical: 0 direction: rtl	glyph-orientation-vertical: 90 unicode-bidi: embed direction: ltr	glyph-orientation-vertical: 90

NOTE:

1. Case (1) has no rotated text. This can occur either because "glyph-orientation-vertical" is set to "0" or because it is set to "auto" and all the characters in the string are the full width variants of the characters. If the orientation of the all glyphs is vertical, then there is no re-ordering of characters. If the "writing-mode" is set to "tb-lr" or "tb-rl" then the "direction" is set to "ltr" and correspondingly, a "writing-mode" set of "bt-lr" or "bt-rl" sets the "direction" to "rtl". Therefore, it is only necessary to explicitly set the "direction" property when it would be different than that set by setting the "writing-mode"; for example, cases (5) through (7).
2. Case (2) can either have the explicit property settings shown in the Table or the "glyph-orientation-vertical" property on the <t> element can have the value "auto" and the English and Hebrew characters can be half-width characters. (Of course, there are not any half-width Hebrew characters in real Unicode.) In this case, the re-ordering of characters comes from the bi-directional characters types that are assigned to each Unicode character: the Roman characters have type "L" for left to right and the Hebrew characters have type "R" for right to left.
3. Cases (5) through (7) all explicitly set the "direction" property to "rtl". This sets the paragraph embedding level for the Unicode Bidi Algorithm to be right to left. Even though the "direction" property is set to "rtl", the ideographic glyphs are not re-ordered because their orientation is not perpendicular to the dominant-baseline.
4. In cases (5) and (6) for the <t-s1> element, the "unicode-bidi" property is set to override even though there is no explicit specification for the "direction" property. The inherited value of the "direction" property (which is "rtl" in this case) is used.
5. In case (7) for the <t-s1> element, the "unicode-bidi" property is set to "embed". It is not necessary to use "bidi-override" because the bi-directional character type for the content of <t-s1> is already "L". (Using the value "bidi-override" would have the same effect as the "embed", however.) The embed resets the embedding level of the content of the <t-s1> to be left to right. Even the "embed" (and the specific setting of the "unicode-bidi" property) is not needed because the bi-directional character type, "L" of the English characters is sufficient to raise the embedding level and cause them to be ordered left to right. Setting the "direction" property to "ltr" is needed if the "unicode-bidi" property is other than "normal" because the inherited value of "direction" is "rtl".

If paired punctuation characters, such as parentheses, had been included in one of the text spans, then these characters may need to be "mirrored" as described in the Unicode Bidi Algorithm. Mirroring a character means reversing the direction the character faces; for example, mirroring a left parenthesis makes it into a right parenthesis. This insures that the mirrored characters always face the text they surround.

If the "glyph-orientation" of the characters to which the glyphs correspond is "90" and the embedding level in which the characters lie is odd, then the paired glyphs need to be mirrored. Alternatively, if the "glyph-orientation" of the characters to which the glyphs correspond is "-90" and the embedding level in which the characters lie is even, then the paired glyphs need to be mirrored. In the example above, parentheses that surround the Latin text would not be mirrored in cases (2), (3) and (8), but would need

to be mirrored in cases (4) through (6). Conversely, parentheses that surround the Hebrew text would not be mirrored in cases (4) through (6), but would need to be mirrored in cases (2), (3), and (8).

Within a string of vertical text, when the value of the "glyph-orientation-vertical" property is "90", then each affected glyph is rotated 90 degrees clockwise. This rotation changes the way the rotated glyph is aligned. The horizontal alignment-point of the rotated glyph is aligned with the appropriate baseline from the vertical baseline-table. The appropriate baseline is the baseline identified by the "alignment-baseline" property of the character(s) that generate the glyph area. For example, if the "alignment-baseline" property is not explicitly specified, Latin glyphs are aligned to the (vertical) "alphabetic" baseline and some Indic glyphs are aligned to the (vertical) "hanging" baseline.

NOTE: If a glyph, such as a ligature or syllabic glyph, is generated from more than one character, then all those characters must have the same value for the "alignment-baseline" property.

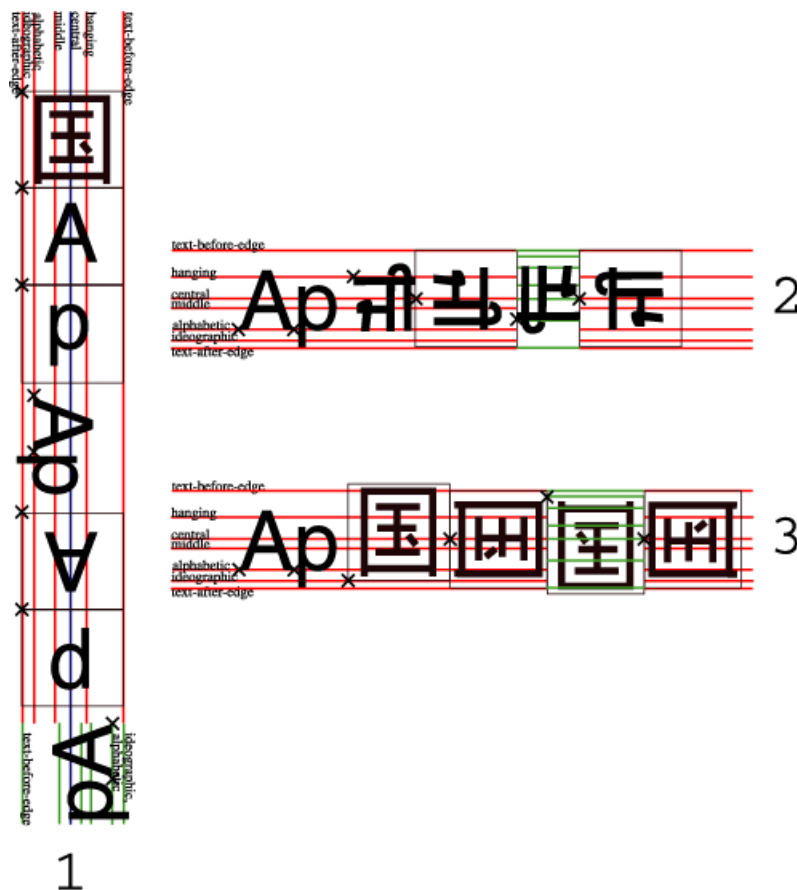
The positions of the (vertical) baselines are chosen to insure that the rotated glyphs will not protrude too far (if at all) outside the line area for the vertical line when the "line-stacking-strategy" property has the value "line-height" or "font-height". In this case, we will say the rotated text is *well aligned* in the vertical line area.

To preserve the property that rotated text in a vertical line is well aligned when the "glyph-orientation-vertical" property value is "-90", the vertical baseline-table must be *reflected* before the rotated text is aligned. Let C be the value of the offset to the "central" baseline in the baseline-table. A baseline-table is reflected by negating every offset in the baseline table (where negating "-N" yields "N") and adding 2 times C to each of the negated offsets. The "central" baseline is defined in § 7.11 – [Area Alignment Properties](#) on page 212.

This action is called "reflecting" because the offset from the original dominant baseline to any baseline in the reflected baseline-table places that baseline on the opposite side of the "central" baseline and the distance from the "central" baseline to that baseline is the same as was from the "central" baseline to that baseline in its original (un-reflected) position. In short, the positions of the baselines are reflected across the "central" baseline.

NOTE: If X is the offset of baseline X and C is the offset of the "central" baseline, then $-X+2*C = C+(C-X)$. $C+(C-X)$ is the offset of the "central" baseline plus the distance between the "central" baseline and baseline X , the baseline being reflected.

Reflecting is necessary, because both the "alphabetic" and the "hanging" baselines are near the outer edges of the vertical line area. If the glyph were simply rotated 180 degrees, then it would stick way out of the intended line area. This is prevented by reflecting the baselines for glyphs that are perpendicular to the dominant baseline and that are rotated 180 degrees from the direction for which that baseline was intended. This last statement applies as much to horizontal baselines as it does to vertical baselines.



The figure illustrates the positioning of rotated and inverted glyphs in both vertical and horizontal writing-modes. The three examples show first some glyphs typical of the writing mode and then some atypical glyphs in each of the possible orientations, 0, 90, 180 and -90 degrees, in that order. The alignment-point for each glyph is shown as a small "x" whose center is at the alignment-point.

Example 1 shows the "tb-rl" vertical writing-mode. It has the ideographic glyph for "country" as its normal glyph and the two letters sequence, "Ap" as the glyphs that are rotated. Note that in the default orientation (0 degrees) and in the inverted orientation, the full width Latin glyphs are used; in the two other orientations, the proportional Latin glyphs are used. There is a small amount of whitespace between the proportional and the full width Latin glyphs. The dominant baseline is the "central" baseline which is shown in blue. The reflected baseline table is shown for the last (-90 degree) rotation. Note that the position of the "central" baseline does not change when the baseline table is reflected. For the inverted glyphs and the glyphs with a -90 degree rotation, the start-edge of the rotated glyph is on the opposite side from where it is in the un-rotated glyph; hence, the alignment-point on that start edge is not on the edge where the font tables normally place it.

Examples 2 and 3 show the "lr-tb" horizontal writing-mode. They have the Latin glyph sequence, "Ap" as their normal glyphs. Example 2 rotates the syllabic Gurmukhi glyph for "ji" and example 3 rotates the ideographic glyph for "country". In example 2, the whole syllabic glyph is rotated as in indivisible unit. For the 90 and -90 degree rotations, the vertical alignment-point, aligning to the "central" baseline, is used in both Examples. Similarly, for the inverted glyph, the baseline table is reflected. For the glyphs with a 90 degree rotation and the inverted glyphs, the start-edge of the rotated glyph is on the opposite side from where it is in the un-rotated glyph; hence, the alignment-point on that start edge is not on the edge where the font tables normally place it.

7.25.1. “direction”

CSS2 Definition:

<i>Value:</i>	ltr rtl inherit
<i>Initial:</i>	ltr
<i>Applies to:</i>	all elements, but see prose
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-direction>.

This property specifies the base writing direction of blocks and the direction of embeddings and overrides (see 'unicode-bidi') for the Unicode bidirectional algorithm. In addition, it specifies the direction of table column layout, the direction of horizontal overflow, and the position of an incomplete last line in a block in case of 'text-align: justify'.

Values for this property have the following meanings:

ltr Left to right direction.

rtl Right to left direction.

For the 'direction' property to have any effect on inline-level elements, the 'unicode-bidi' property's value must be 'embed' or 'override'.

NOTE: The 'direction' property, when specified for table column elements, is not inherited by cells in the column since columns don't exist in the document tree. Thus, CSS cannot easily capture the "dir" attribute inheritance rules described in [HTML40], section 11.3.2.1.

XSL modifications to the CSS definition:

- The specific use of "direction" and "unicode-bidi" on inline objects is to set the inline-progression-direction to be used by the Unicode bidi algorithm. This direction may override the inline-progression-direction determined by the current writing-mode and the implicit direction determined by the Unicode bidirectional algorithm.
- To insure consistency with the "writing-mode" property, the "direction" property is initialized to the value that sets the same inline-progression-direction as is set by the "writing-mode" property whenever that "writing-mode" property sets that direction. If the "direction" property is explicitly specified on the same formatting object the value of the "direction" property will override the inline-progression-direction set by the "writing-mode".
- This property only has an effect on text in which the orientation of the glyphs is perpendicular to the inline-progression-direction. Therefore, vertical ideographic text with the initial value for "glyph-orientation-vertical" is not affected by this property; vertical text for which the "glyph-orientation-vertical" property has the value of "90" or "-90" degrees is affected.

NOTE: When the inline-progression-direction is "tb", as is typical for vertical text, then this corresponds to a "lr" inline-progression-direction for text with a glyph-orientation of '90' degrees and an "rl" inline-progression-direction

for text with a glyph-orientation of "-90" degrees.

- The "writing-mode" property is used on formatting objects that define blocks that generate reference-areas, including inline-containers. It establishes both the block-progression-direction and the inline-progression-direction. The "direction" property only changes the inline-progression-direction and is used primarily for formatting objects that generate inline areas that are not also reference areas. Use of the "direction" property for other formatting objects is deprecated in this specification.
- When mapping CSS to XSL, the XSL "writing-mode" property should be used rather than the "direction" property for all block-level directionality control. XSL's "writing-mode" should also be used for any inline-container or block-container objects. The "direction" property should be used only for control/overrides of the Unicode Bidi Algorithm on bidi-override formatting objects.

7.25.2. “font-height-override-after”

XSL Definition:

<i>Value:</i>	use-font-metrics <length> inherit
<i>Initial:</i>	use-font-metrics
<i>Applies to:</i>	fo:block, fo:character, fo:leader
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to font's em-height
<i>Media:</i>	visual

Values have the following meanings:

use-font-metrics

Uses a value for the height of the font below the baseline obtained from the actual font used for fo:character and for fo:leader when the *leader-pattern* has the value "use-content", the nominal font in all other cases.

<length>

Replaces the height value found in the font.

Specifies the height to be used for the descent below the font's reference-position (baseline).

7.25.3. “font-height-override-before”

XSL Definition:

<i>Value:</i>	use-font-metrics <length> inherit
<i>Initial:</i>	use-font-metrics
<i>Applies to:</i>	fo:block, fo:character, fo:leader
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to font's em-height
<i>Media:</i>	visual

Values have the following meanings:

use-font-metrics

Uses a value for the height of the font above the baseline obtained from the actual font used for fo:character and for fo:leader when the *leader-pattern* has the value "use-content", the nominal font in all other cases.

<length>

Replaces the height value found in the font.

Specifies the height to be used for the ascent above the font's reference-position (baseline).

7.25.4. "glyph-orientation-horizontal"

XSL Definition:

<i>Value:</i>	<angle> inherit
<i>Initial:</i>	0
<i>Applies to:</i>	fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<angle>

The angle is restricted to a range of -360 to +360 in 90-degree increments.

A value of "0" indicates that all glyphs are set with the top of the glyphs toward the top of the reference-area. The top of the reference-area is defined by the reference-area's *reference-orientation*.

A value of "90" indicates a rotation of 90-degrees clockwise from the "0" orientation.

The angle value is computed modulo 360; thus a value of "-90" or a value of "270" indicates a rotation of 90-degrees counter-clockwise from the "0" orientation.

This property specifies the orientation of glyphs relative to the path direction specified by the 'writing-mode'. This property is applied only to text written in a horizontal writing-mode.

The value of this property affects both the alignment and width of the glyph-areas generated for the affected glyphs. If a glyph is oriented so that it is not perpendicular to the dominant-baseline, then the vertical alignment-point of the rotated glyph is aligned with the alignment-baseline appropriate to that glyph. The baseline to which the rotated glyph is aligned is the (horizontal) baseline identified by the "alignment-baseline" for the script to which the glyph belongs. The width of the glyph-area is determined from the vertical width font characteristic for the glyph.

7.25.5. “glyph-orientation-vertical”

XSL Definition:

<i>Value:</i>	auto <angle> inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:character
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

auto

- Fullwidth ideographic and fullwidth Latin text (excluding ideographic punctuation) will be set with a glyph-orientation of 0.

Ideographic punctuation and other ideographic characters having alternate horizontal and vertical forms will use the vertical form of the glyph.

- Text which is not fullwidth will be set with a glyph-orientation of 90.

This reorientation rule applies only to the first-level non-ideographic text. All further embedding of writing-modes or bidi processing will be based on the first-level rotation.

NOTE:

- This is equivalent to having set the non-ideographic text string horizontally honoring the bidi-rule, then rotating the resultant sequence of inline-areas (one area for each change of glyph direction) 90-degrees clockwise.

It should be noted that text set in this "rotated" manner may contain ligatures or other glyph combining and reordering common to the language and script. (This "rotated" presentation form does not disable auto-ligature formation or similar context-driven variations.)

- The determination of which characters should be auto-rotated may vary across User Agents. The determination is based on a complex interaction between country, language, script, character properties, font, and character context. It is suggested that one consult the Unicode TR 11 and the various JIS or other national standards.

<angle>

The angle is restricted to a range of -360 to +360 in 90-degree increments.

A value of "0" indicates that all glyphs are set with the top of the glyphs toward the top of the reference-area. The top of the reference-area is defined by the reference-area's *reference-orientation*.

A value of "90" indicates a rotation of 90-degrees clockwise from the "0" orientation.

The angle value is computed modulo 360; thus a value of "-90" or a value of "270" indicates a rotation of 90-degrees counter-clockwise from the "0" orientation.

This property specifies the orientation of glyphs relative to the path direction specified by the writing-mode. This property is applied only text written with an inline-progression-direction

top-to-bottom or bottom-to-top.

Its most common usage is to differentiate between the preferred orientation of alphabetic text in vertically written Japanese documents (`glyph-orientation="auto"`) vs. the orientation of alphabetic text in western signage and advertising (`glyph-orientation="0"`).

The value of this property affects both the alignment and width of the glyph-areas generated for the affected glyphs. If a glyph is oriented so that it is perpendicular to the dominant-baseline, then the horizontal alignment-point of the rotated glyph is aligned with the alignment-baseline appropriate to that glyph. The baseline to which the rotated glyph is aligned is the (vertical) baseline identified by the "alignment-baseline" for the script to which the glyph belongs. The width of the glyph-area is determined from the horizontal width font characteristic for the glyph.

7.25.6. “unicode-bidi”

CSS2 Definition:

<i>Value:</i>	normal embed bidi-override inherit
<i>Initial:</i>	normal
<i>Applies to:</i>	all elements, but see prose
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-unicode-bidi>.

Values have the following meanings:

normal

The element does not open an additional level of embedding with respect to the bidirectional algorithm.

For inline-level elements, implicit reordering works across element boundaries.

embed

If the element is inline-level, this value opens an additional level of embedding with respect to the bidirectional algorithm. The direction of this embedding level is given by the 'direction' property. Inside the element, reordering is done implicitly. This corresponds to adding a LRE (U+202A; for 'direction: ltr') or RLE (U+202B; for 'direction: rtl') at the start of the element and a PDF (U+202C) at the end of the element.

bidi-override

If the element is inline-level or a block-level element that contains only inline-level elements, this creates an override. This means that inside the element, reordering is strictly in sequence according to the 'direction' property; the implicit part of the bidirectional algorithm is ignored. This corresponds to adding a LRO (U+202D; for 'direction: ltr') or RLO (U+202E; for 'direction: rtl') at the start of the element and a PDF (U+202C) at the end of the element.

The final order of characters in each block-level element is the same as if the bidi control codes had

been added as described above, markup had been stripped, and the resulting character sequence had been passed to an implementation of the Unicode bidirectional algorithm for plain text that produced the same line-breaks as the styled text. In this process, non-textual entities such as images are treated as neutral characters, unless their 'unicode-bidi' property has a value other than 'normal', in which case they are treated as strong characters in the 'direction' specified for the element.

Please note that in order to be able to flow inline boxes in a uniform direction (either entirely left-to-right or entirely right-to-left), more inline boxes (including anonymous inline boxes) may have to be created, and some inline boxes may have to be split up and reordered before flowing.

Because the Unicode algorithm has a limit of 15 levels of embedding, care should be taken not to use 'unicode-bidi' with a value other than 'normal' unless appropriate. In particular, a value of 'inherit' should be used with extreme caution. However, for elements that are, in general, intended to be displayed as blocks, a setting of 'unicode-bidi: embed' is preferred to keep the element together in case display is changed to inline.

XSL modifications to the CSS definition:

The phrasing of the first paragraph of the general description (following the value breakouts) should read "The final order of presentation of the characters...".

In Unicode 3.0, the Unicode Consortium has increased the limit of the levels of embedding to 61 (definition BD2 in [\[UNICODE TR9\]](#)).

Fallback:

If it is not possible to present the characters in the correct order, then the User Agent should display either and "unsupported character glyph" or display some indication that the content cannot be correctly rendered.

7.25.7. “writing-mode”

XSL Definition:

<i>Value:</i>	lr-tb rl-tb tb-rl lr rl tb inherit
<i>Initial:</i>	lr-tb
<i>Applies to:</i>	see prose
<i>Inherited:</i>	yes (see prose)
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

NOTE: This version of the writing-mode property covers the base writing-modes that are used as the official languages of the United Nations. For information regarding additional writing-modes, please see the "Internationalization Appendix".

Values have the following meanings:

lr-tb

Inline components and text within a line are written left-to-right. Lines and blocks are placed

top-to-bottom.

NOTE: Typically, this is the writing-mode for normal "alphabetic" text.

Establishes the following directions:

- inline-progression-direction to left-to-right
If any right-to-left reading characters are present in the text, the inline-progression-direction for glyph-areas may be further modified by the Unicode bidi algorithm.
- block-progression-direction to top-to-bottom
- shift-direction to bottom-to-top

rl-tb

Inline components and text within a line are written right-to-left. Lines and blocks are placed top-to-bottom.

NOTE: Typically, this writing mode is used in Arabic and Hebrew text.

Establishes the following directions:

- inline-progression-direction to right-to-left
If any left-to-right reading characters or numbers are present in the text, the inline-progression-direction for glyph-areas may be further modified by the Unicode bidi algorithm.
- block-progression-direction to top-to-bottom
- shift-direction to bottom-to-top

tb-rl

Inline components and text within a line are written top-to-bottom. Lines and blocks are placed right-to-left.

NOTE: Typically, this writing mode is used in Chinese and Japanese text.

Establishes the following directions:

- inline-progression-direction to top-to-bottom
- block-progression-direction to right-to-left
- shift-direction to left-to-right

lr

Shorthand for lr-tb.

rl

Shorthand for rl-tb.

tb

Shorthand for tb-rl.

The "writing-mode" property is applied (processed and converted to the three direction traits) only on formatting objects that set up a reference-area (for XSL these are: fo:simple-page-master, fo:region-*, fo:table, fo:block-container, and fo:inline-container. Each value of writing-mode sets all three of the direction traits indicated in each of the value descriptions above on the reference-area. (See the area model for a description of the direction traits and their usage.)

- When "writing-mode" is applied to the simple-page-master, it is used to determine the placement of the five regions on the master.
- When "writing-mode" is applied to the fo:*-region, it defines the column-progression within each region. The inline-progression-direction is used to determine the stacking direction for columns (and the default flow order of text from column-to-column).
- To change the "writing-mode" within an fo:flow or fo:static-content, either the fo:block-container or the fo:inline-container, as appropriate, should be used.

If one only wishes to change the inline-progression-direction to override the Unicode bidi-rule, one need not use an fo:inline-container. Instead, one may use the "direction" property on the fo:bidi-override.

- When "writing-mode" is applied to the fo:table, it controls the layout of the rows and columns. Table-rows use the block-progression-direction as the row-stacking direction. The inline-progression-direction is used to determine the stacking direction for columns (and cell order within the row).

Implementations must support at least one of the "writing-mode" values defined in this Recommendation.

7.26. Miscellaneous Properties

7.26.1. "content-type"

XSL Definition:

<i>Value:</i>	<string> auto
<i>Initial:</i>	auto
<i>Applies to:</i>	fo:external-graphic, fo:instream-foreign-object
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

This property specifies the content-type and may be used by a User Agent to select a rendering processor for the object.

Values for this property have the following meanings:

auto

No identification of the content-type. The User Agent may determine it by "sniffing" or by other means.

<string>

A specification of the content-type in terms of either a mime-type or a namespace.

A mime-type specification has the form "content-type:" followed by a mime content-type, e.g., content-type="content-type:xml/svg".

A namespace specification has the form "namespace-prefix:" followed by a declared namespace prefix, e.g., content-type="namespace-prefix:svg". If the namespace prefix is null, the content-type refers to the default namespace.

7.26.2. “id”

XSL Definition:

<i>Value:</i>	<id>
<i>Initial:</i>	see prose
<i>Applies to:</i>	all formatting objects
<i>Inherited:</i>	no, see prose
<i>Percentages:</i>	N/A
<i>Media:</i>	all

Values have the following meanings:

<id>

An identifier unique within all objects in the result tree with the fo: namespace. It allows references to this formatting object by other objects.

The "inherit" value is not allowed on this property.

The initial value of this property is random and unique identifier. The algorithm to generate this identifier is system-dependent.

7.26.3. “provisional-label-separation”

XSL Definition:

<i>Value:</i>	<length> inherit
<i>Initial:</i>	6.0pt
<i>Applies to:</i>	fo:list-block
<i>Inherited:</i>	yes
<i>Percentages:</i>	refer to width of the containing box
<i>Media:</i>	visual

Values have the following meanings:

<length>

Specifies the provisional distance between the end of the list-item-label and the start of the list-item-body. The value is not directly used during formatting, but is used in the computation of the value of the label-end variable.

NOTE: label-end() = width of the content-rectangle of the reference-area into which the list-block is placed - (the value of the provisional-distance-between-starts + the value of the start-indent - the value of the

provisional-label-separation) of the closest ancestor fo:list-block.

7.26.4. “provisional-distance-between-starts”

XSL Definition:

<i>Value:</i>	<length> inherit
<i>Initial:</i>	24.0pt
<i>Applies to:</i>	fo:list-block
<i>Inherited:</i>	yes
<i>Percentages:</i>	refer to width of the containing box
<i>Media:</i>	visual

Values have the following meanings:

<length>

Specifies the provisional distance between the start-indent of the list-item-label and the start-indent of the list-item-body. The value is not directly used during formatting, but is used in the computation of the value of the body-start variable.

NOTE: body-start() = the value of the start-indent + the value of the provisional-distance-between-starts of the closest ancestor fo:list-block.

7.26.5. “ref-id”

XSL Definition:

<i>Value:</i>	<idref> inherit
<i>Initial:</i>	none, value required
<i>Applies to:</i>	fo:page-number-citation
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	all

Values have the following meanings:

<idref>

The "id" of an object in the formatting object tree.

Reference to the object having the specified unique identifier.

7.26.6. “score-spaces”

XSL Definition:

<i>Value:</i>	true false inherit
<i>Initial:</i>	true

<i>Applies to:</i>	fo:bidirectional-override, fo:character, fo:initial-property-set, fo:page-number, fo:page-number-citation
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

true

Text-decoration will be applied to spaces

false

Text-decoration will not be applied to spaces

Specifies whether the text-decoration property shall be applied to spaces.

7.26.7. “src”

XSL Definition:

<i>Value:</i>	<uri-specification> inherit
<i>Initial:</i>	none, value required
<i>Applies to:</i>	fo:external-graphic
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<uri-specification>

Specifies the URI reference to locate an external resource such as image/graphic data to be included as the content of this object, or color-profile data.

7.26.8. “visibility”

CSS2 Definition:

<i>Value:</i>	visible hidden collapse inherit
<i>Initial:</i>	visible
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visufx.html#propdef-visibility>.

The 'visibility' property specifies whether the boxes generated by an element are rendered. Invisible boxes still affect layout (set the 'display' property to 'none' to suppress box generation altogether). Values have the following meanings:

visible

The generated box is visible.

hidden

The generated box is invisible (fully transparent), but still affects layout.

collapse

Please consult the section on dynamic row and column effects in tables. If used on elements other than rows or columns, "collapse" has the same meaning as "hidden".

This property may be used in conjunction with scripts to create dynamic effects.

XSL modifications to the CSS definition:

Changed initial value to visible; (it is "inherit" in CSS).

7.26.9. “z-index”

CSS2 Definition:

Value: auto | <integer> | inherit

Initial: auto

Applies to: positioned elements

Inherited: no

Percentages: N/A

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-z-index>.

For a positioned box, the "z-index" property specifies:

1. The stack level of the box in the current stacking context.
2. Whether the box establishes a local stacking context.

Values have the following meanings:

auto

The stack level of the generated box in the current stacking context is the same as its parent's box. The box does not establish a new local stacking context.

<integer>

This integer is the stack level of the generated box in the current stacking context. The box also establishes a local stacking context in which its stack level is "0".

This example [see the CSS specification] demonstrates the notion of transparency. The default behavior of a box is to allow boxes behind it to be visible through transparent areas in its content. In the example, each box transparently overlays the boxes below it. This behavior can be overridden by using one of the existing background properties.

7.27. Shorthand Properties

The following properties are all shorthand properties. Shorthands are only included in the highest XSL conformance level: "complete".

7.27.1. "background"

CSS2 Definition:

<i>Value:</i>	[<background-color> <background-image> <background-repeat> <background-attachment> <background-position>] inherit
<i>Initial:</i>	not defined for shorthand properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	allowed on 'background-position'
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/colors.html#propdef-background>.

The "background" property is a shorthand property for setting the individual background properties (i.e., background-color, background-image, background-repeat, background-attachment and background-position) at the same place in the stylesheet.

The "background" property first sets all the individual background properties to their initial values, then assigns explicit values given in the declaration.

7.27.2. "background-position"

CSS2 Definition:

<i>Value:</i>	[[<percentage> <length>]{1,2}] [[top center bottom] [left center right]] inherit
<i>Initial:</i>	0% 0%
<i>Applies to:</i>	block-level and replaced elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to the size of the box itself
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/colors.html#propdef-background-position>.

If a "background-image" has been specified, this property specifies its initial position.

<percentage> <percentage>

With a value pair of 0% 0%, the upper left corner of the image is aligned with the upper left corner of the box's padding edge. A value pair of 100% 100% places the lower right corner of the image in the lower right corner of padding area. With a value pair of 14% 84%, the point 14% across and 84% down the image is to be placed at the point 14% across and 84% down the padding area.

<length> <length>

With a value pair of 2cm 2cm, the upper left corner of the image is placed 2cm to the right and 2cm below the upper left corner of the padding area.

top left and left top

Same as 0% 0%.

top, top center, and center top

Same as 50% 0%.

right top and top right

Same as 100% 0%.

left, left center, and center left

Same as 0% 50%.

center and center center

Same as 50% 50%.

right, right center, and center right

Same as 100% 50%.

bottom left and left bottom

Same as 0% 100%.

bottom, bottom center, and center bottom

Same as 50% 100%.

bottom right and right bottom

Same as 100% 100%.

If only one percentage or length value is given, it sets the horizontal position only, the vertical position will be 50%. If two values are given, the horizontal position comes first. Combinations of length and percentage values are allowed, (e.g., 50% 2cm). Negative positions are allowed. Keywords cannot be combined with percentage values or length values (all possible combinations are given above).

If the background image is fixed within the viewport (see the "background-attachment" property), the image is placed relative to the viewport instead of the elements padding area.

XSL modifications to the CSS definition:

The CSS property shall be treated as a shorthand by XSL and maps as follows:

<percentage>

background-position-horizontal="<percentage>"

background-position-vertical="50% "

<percentage1> <percentage2>

background-position-horizontal="<percentage1>"

background-position-vertical="<percentage2>"

<length>

background-position-horizontal="<length>"

background-position-vertical="50% "

<length1> <length2>

background-position-horizontal="<length1>"

background-position-vertical="<length2>"

<length> <percentage>

background-position-horizontal="<length>"

background-position-vertical="<percentage>"

<percentage> <length>

background-position-horizontal="<percentage>"

background-position-vertical="<length>"

top left and left top

background-position-horizontal="0% "

background-position-vertical="0% "

top, top center, and center top

background-position-horizontal="50% "

background-position-vertical="0% "

right top and top right

background-position-horizontal="100% "

background-position-vertical="0% "

left, left center, and center left

background-position-horizontal="0% "

background-position-vertical="50% "

center and center center

background-position-horizontal="50% "

background-position-vertical="50% "

right, right center, and center right

background-position-horizontal="100% "

background-position-vertical="50% "

bottom left and left bottom

background-position-horizontal="0% "

background-position-vertical="100% "

bottom, bottom center, and center bottom

background-position-horizontal="50% "

background-position-vertical="100% "

bottom right and right bottom

background-position-horizontal="100% "

background-position-vertical="100% "

7.27.3. “border”**CSS2 Definition:***Value:* [<border-width> || <border-style> || <color>] | inherit*Initial:* see individual properties*Applies to:* all elements*Inherited:* no*Percentages:* N/A*Media:* visualCSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border>.

The "border" property is a shorthand property for setting the same width, color, and style for all four borders, top, bottom, left, and right, of a box. Unlike the shorthand "margin" and "padding" properties, the "border" property cannot set different values on the four borders. To do so, one or more of the other border properties must be used.

XSL modifications to the CSS definition:

Refer to § 5.3.1 – [Border and Padding Properties](#) on page 42 for information on the precedence order of properties.

7.27.4. “border-bottom”

CSS2 Definition:

<i>Value:</i>	[<border-top-width> <border-style> <color>] inherit
<i>Initial:</i>	see individual properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-bottom>.

A shorthand property for setting the width, style, and color of the bottom border of a block-area or inline-area.

7.27.5. “border-color”

CSS2 Definition:

<i>Value:</i>	<color>{ 1,4 } transparent inherit
<i>Initial:</i>	see individual properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-color>.

The 'border-color' property sets the color of the four borders. Values have the following meanings:

transparent

The border is transparent (though it may have width).

<color>

Any valid color specification.

The "border-color" property can have from one to four values, and the values are set on the different sides as for "border-width".

If an element's border color is not specified with a "border" property, user agents must use the value of the element's "color" property as the computed value for the border color.

XSL modifications to the CSS definition:

See the 'border-width' property for a description of how this property is interpreted when one through four values are provided.

7.27.6. “border-left”

CSS2 Definition:

<i>Value:</i>	[<border-top-width> <border-style> <color>] inherit
<i>Initial:</i>	see individual properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-left>.

A shorthand property for setting the width, style, and color of the left border of a block-area or inline-area.

7.27.7. “border-right”

CSS2 Definition:

<i>Value:</i>	[<border-top-width> <border-style> <color>] inherit
<i>Initial:</i>	see individual properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-right>.

A shorthand property for setting the width, style, and color of the right border of a block-area or inline-area.

7.27.8. “border-style”

CSS2 Definition:

<i>Value:</i>	<border-style>{ 1,4 } inherit
<i>Initial:</i>	see individual properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-style>.

The "border-style" property sets the style of the four borders.

It can have from one to four values, and the values are set on the different sides.

XSL modifications to the CSS definition:

See the 'border-width' property for a description of how this property is interpreted when one through four values are provided.

7.27.9. “border-spacing”

CSS2 Definition:

Value: <length> <length>? | inherit
Initial: Opt
Applies to: table
Inherited: yes
Percentages: N/A
Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/tables.html#propdef-border-spacing>.

<length>

The lengths specify the distance that separates adjacent cell borders. If one length is specified, it gives both the horizontal and vertical spacing. If two are specified, the first gives the horizontal spacing and the second the vertical spacing. Lengths may not be negative.

In the separate borders model, each cell has an individual border. The "border-spacing" property specifies the distance between the borders of adjacent cells. This space is filled with the background of the table element. Rows, columns, row groups, and column groups cannot have borders (i.e., user agents must ignore the border properties for those elements).

XSL modifications to the CSS definition:

The CSS property shall be treated as a shorthand by XSL and maps as follows:

If one value is specified the "border-separation.block-progression-direction" and "border-separation.inline-progression-direction" are both set to that value.

If two values are specified the "border-separation.block-progression-direction" is set to the second value and "border-separation.inline-progression-direction" are both set to the first value.

7.27.10. “border-top”

CSS2 Definition:

Value: [<border-top-width> || <border-style> || <color>] | inherit
Initial: see individual properties
Applies to: all elements
Inherited: no

Percentages: N/A
Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-top>.

A shorthand property for setting the width, style, and color of the top border of a block-area or inline-area.

7.27.11. “border-width”

CSS2 Definition:

Value: <border-width>{ 1,4 } | inherit
Initial: see individual properties
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-border-width>.

This property is a shorthand property for setting "border-top-width", "border-right-width", "border-bottom-width", and "border-left-width" at the same place in the stylesheet.

If there is only one value, it applies to all sides. If there are two values, the top and bottom borders are set to the first value and the right and left are set to the second. If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third. If there are four values, they apply to the top, right, bottom, and left, respectively.

7.27.12. “cue”

CSS2 Definition:

Value: <cue-before> || <cue-after> | inherit
Initial: not defined for shorthand properties
Applies to: all elements
Inherited: no
Percentages: N/A
Media: aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-cue>.

7.27.13. “font”

CSS2 Definition:

<i>Value:</i>	[[<font-style> <font-variant> <font-weight>]? <font-size> [/ <line-height>]? <font-family>] caption icon menu message-box small-caption status-bar inherit
<i>Initial:</i>	see individual properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/fonts.html#propdef-font>.

The "font" property is, except as described below, a shorthand property for setting "font-style", "font-variant", "font-weight", "font-size", "line-height", and "font-family", at the same place in the stylesheet. The syntax of this property is based on a traditional typographical shorthand notation to set multiple properties related to fonts.

All font-related properties are first reset to their initial values, including those listed in the preceding paragraph plus "font-stretch" and "font-size-adjust". Then, those properties that are given explicit values in the "font" shorthand are set to those values. For a definition of allowed and initial values, see the previously defined properties. For reasons of backward compatibility, it is not possible to set "font-stretch" and "font-size-adjust" to other than their initial values using the "font" shorthand property; instead, set the individual properties.

The following [first six] values refer to system fonts:

caption

The font used for captioned controls (e.g., buttons, drop-downs, etc.).

icon

The font used to label icons.

menu

The font used in menus (e.g., dropdown menus and menu lists).

message-box

The font used in dialog boxes.

small-caption

The font used for labeling small controls.

status-bar

The font used in window status bars.

System fonts may only be set as a whole; that is, the "font-family", "size", "weight", "style", etc. are

all set at the same time. These values may then be altered individually if desired. If no font with the indicated characteristics exists on a given platform, the user agent should either intelligently substitute (e.g., a smaller version of the "caption" font might be used for the "smallcaption" font), or substitute a user agent default font. As for regular fonts, if, for a system font, any of the individual properties are not part of the operating system's available user preferences, those properties should be set to their initial values.

That is why this property is "almost" a shorthand property: system fonts can only be specified with this property, not with "font-family" itself, so "font" allows authors to do more than the sum of its sub-properties. However, the individual properties such as "font-weight" are still given values taken from the system font, which can be independently varied.

XSL modifications to the CSS definition:

In XSL the "font" property is a pure shorthand property. System font characteristics, such as font-family, and font-size, may be obtained by the use of the "system-font" function in the expression language.

7.27.14. "margin"

CSS2 Definition:

<i>Value:</i>	<margin-width>{ 1,4 } inherit
<i>Initial:</i>	not defined for shorthand properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-margin>.

A shorthand property for setting margin-top, margin-right, margin-bottom, and margin-left of a block-area or inline-area.

If there is only one value, it applies to all sides. If there are two values, the top and bottom margins are set to the first value and the right and left margins are set to the second. If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third. If there are four values, they apply to the top, right, bottom, and left, respectively.

XSL modifications to the CSS definition:

- Margin is provided for compatibility with CSS.
- Details on the mapping of CSS "margin" properties for XSL are given in [Property Refinement / Resolution](#) — § 5 on page 39.

7.27.15. “padding”

CSS2 Definition:

<i>Value:</i>	<padding-width>{ 1,4 } inherit
<i>Initial:</i>	not defined for shorthand properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/box.html#propdef-padding>.

A shorthand property for setting padding-top, padding-bottom, padding-left, and padding-right of a block-area or inline-area.

If there is only one value, it applies to all sides. If there are two values, the top and bottom paddings are set to the first value and the right and left paddings are set to the second. If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third. If there are four values, they apply to the top, right, bottom, and left, respectively.

The surface color or image of the padding area is specified via the "background" property.

7.27.16. “page-break-after”

CSS2 Definition:

<i>Value:</i>	auto always avoid left right inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	block-level elements, list-item, and table-row.
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/page.html#propdef-page-break-after>.

Values for these properties have the following meanings:

auto

Neither force nor forbid a page break before (after, inside) the generated box.

always

Always force a page break before (after) the generated box.

avoid

Avoid a page break before (after, inside) the generated box.

left

Force one or two page breaks before (after) the generated box so that the next page is formatted as a left page.

right

Force one or two page breaks before (after) the generated box so that the next page is formatted as a right page.

A potential page break location is typically under the influence of the parent element's 'page-break-inside' property, the 'page-break-after' property of the preceding element, and the 'page-break-before' property of the following element. When these properties have values other than 'auto', the values 'always', 'left', and 'right' take precedence over 'avoid'. See the section on allowed page breaks for the exact rules on how these properties may force or suppress a page break.

XSL modifications to the CSS definition:

The CSS property shall be treated as a shorthand by XSL and maps as follows:

auto

break-after = "auto"
keep-with-next = "auto"

always

break-after = "page"
keep-with-next = "auto"

avoid

break-after = "auto"
keep-with-next = "always"

left

break-after = "even-page"
keep-with-next = "auto"

right

break-after = "odd-page"
keep-with-next = "auto"

7.27.17. “page-break-before”

CSS2 Definition:

Value: auto | always | avoid | left | right | inherit
Initial: auto

Applies to: block-level elements, list-item, and table-row.

Inherited: no

Percentages: N/A

Media: visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/page.html#propdef-page-break-before>.

Values for these properties have the following meanings:

auto

Neither force nor forbid a page break before (after, inside) the generated box.

always

Always force a page break before (after) the generated box.

avoid

Avoid a page break before (after, inside) the generated box.

left

Force one or two page breaks before (after) the generated box so that the next page is formatted as a left page.

right

Force one or two page breaks before (after) the generated box so that the next page is formatted as a right page.

A potential page break location is typically under the influence of the parent element's 'page-break-inside' property, the 'page-break-after' property of the preceding element, and the 'page-break-before' property of the following element. When these properties have values other than 'auto', the values 'always', 'left', and 'right' take precedence over 'avoid'. See the section on allowed page breaks for the exact rules on how these properties may force or suppress a page break.

XSL modifications to the CSS definition:

The CSS property shall be treated as a shorthand by XSL and maps as follows:

auto

break-before = "auto"

keep-with-previous = "auto"

always

break-before = "page"

keep-with-previous = "auto"

avoid

break-before = "auto"

keep-with-previous = "always"

left

break-before = "even-page"

keep-with-previous = "auto"

right

break-before = "odd-page"

keep-with-previous = "auto"

7.27.18. “page-break-inside”

CSS2 Definition:

<i>Value:</i>	avoid auto inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	block-level elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/page.html#propdef-page-break-inside>.

NOTE: The CSS definition for page-break-inside was shared with the definitions of page-break-before and page-break-after. The text here has been edited to include only the value choices valid for page-break-inside and to remove the before/after/inside triplet.

Values for this property have the following meanings:

auto

Neither force nor forbid a page break inside the generated box.

avoid

Avoid a page break inside the generated box.

A potential page break location is typically under the influence of the parent element's 'page-break-inside' property, the 'page-break-after' property of the preceding element, and the 'page-break-before' property of the following element. When these properties have values other than 'auto', values 'always', 'left', and 'right' take precedence over 'avoid'. See the section on allowed page breaks for the exact rules on how these properties may force or suppress a page break.

XSL modifications to the CSS definition:

XSL treats this as a shorthand and maps it as follows.

auto

keep-together = "auto"

avoid

keep-together = "always"

7.27.19. “pause”

CSS2 Definition:

<i>Value:</i>	[<time> <percentage>]{1,2} inherit
<i>Initial:</i>	depends on user agent
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	see descriptions of 'pause-before' and 'pause-after'
<i>Media:</i>	aural

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/aural.html#propdef-pause>.

7.27.20. “position”

CSS2 Definition:

<i>Value:</i>	static relative absolute fixed inherit
<i>Initial:</i>	static
<i>Applies to:</i>	all elements, but not to generated content
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visuren.html#propdef-position>.

Values have the following meanings:

static

The box is a normal box, laid out according to the normal flow. The "left" and "top" properties do not apply.

relative

The box's position is calculated according to the normal flow (this is called the position in normal flow). Then the box is offset relative to its normal position. When a box B is relatively positioned, the position of the following box is calculated as though B were not offset.

absolute

The box's position (and possibly size) is specified with the "left", "right", "top", and "bottom" properties. These properties specify offsets with respect to the box's containing block. Absolutely positioned boxes are taken out of the normal flow. This means they have no impact on the layout of later siblings. Also, though absolutely positioned boxes have margins,

they do not collapse with any other margins.

fixed

The box's position is calculated according to the "absolute" model, but in addition, the box is fixed with respect to some reference. In the case of continuous media, the box is fixed with respect to the viewport (and doesn't move when scrolled). In the case of paged media, the box is fixed with respect to the page, even if that page is seen through a viewport (in the case of a print-preview, for example). Authors may wish to specify "fixed" in a media-dependent way. For instance, an author may want a box to remain at the top the viewport on the screen, but not at the top of each printed page.

Specifies the positioning scheme to be used.

XSL modifications to the CSS definition:

The CSS property shall be treated as a shorthand by XSL and maps as follows:

static

relative-position="static"

absolute-position="auto"

relative

relative-position="relative"

absolute-position="auto"

absolute

relative-position="static"

absolute-position="absolute"

fixed

relative-position="static"

absolute-position="fixed"

7.27.21. “size”

CSS2 Definition:

<i>Value:</i>	<length>{ 1,2 } auto landscape portrait inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	the page context
<i>Inherited:</i>	N/A [XSL:no, is optional]
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/page.html#propdef-size>.

This property specifies the size and orientation of a page box.

The size of a page box may either be "absolute" (fixed size) or "relative" (scalable, i.e., fitting available sheet sizes). Relative page boxes allow user agents to scale a document and make optimal use of the target size.

[The first] Three values for the 'size' property create a relative page box:

auto

The page box will be set to the size and orientation of the target sheet.

landscape

Overrides the target's orientation. The page box is the same size as the target, and the longer sides are horizontal.

portrait

Overrides the target's orientation. The page box is the same size as the target, and the shorter sides are horizontal.

<length>

Length values for the "size" property create an absolute page box. If only one length value is specified, it sets both the width and height of the page box (i.e., the box is a square). Since the page box is the initial containing block, percentage values are not allowed for the "size" property.

User agents may allow users to control the transfer of the page box to the sheet (e.g., rotating an absolute page box that's being printed).

- Rendering page boxes that do not fit a target sheet

If a page box does not fit the target sheet dimensions, the user agent may choose to:

- Rotate the page box 90 degrees if this will make the page box fit.
- Scale the page to fit the target.

The user agent should consult the user before performing these operations.

- Positioning the page box on the sheet

When the page box is smaller than the target size, the user agent is free to place the page box anywhere on the sheet. However, it is recommended that the page box be centered on the sheet since this will align double-sided pages and avoid accidental loss of information that is printed near the edge of the sheet.

XSL modifications to the CSS definition:

This is treated as a CSS shorthand property that is mapped to XSL's "page-height" and "page-width" properties.

7.27.22. “vertical-align”

CSS2 Definition:

<i>Value:</i>	baseline middle sub super text-top text-bottom <percentage> <length> top bottom inherit
<i>Initial:</i>	baseline
<i>Applies to:</i>	inline-level and 'table-cell' elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to the 'line-height' of the element itself
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/visudet.html#propdef-vertical-align>.

This property affects the vertical positioning inside a line box of the boxes generated by an inline-level element. The following values only have meaning with respect to a parent inline-level element, or to a parent block-level element, if that element generates anonymous inline boxes; they have no effect if no such parent exists.

NOTE: Values of this property have slightly different meanings in the context of tables. Please consult the section on table height algorithms for details.

Values have the following meanings:

baseline

Align the baseline of the box with the baseline of the parent box. If the box doesn't have a baseline, align the bottom of the box with the parent's baseline.

middle

Align the vertical midpoint of the box with the baseline of the parent box plus half the x-height of the parent.

sub

Lower the baseline of the box to the proper position for subscripts of the parent's box. (This value has no effect on the font size of the element's text.)

super

Raise the baseline of the box to the proper position for superscripts of the parent's box. (This value has no effect on the font size of the element's text.)

text-top

Align the top of the box with the top of the parent element's font.

text-bottom

Align the bottom of the box with the bottom of the parent element's font.

top

Align the top of the box with the top of the line box.

bottom

Align the bottom of the box with the bottom of the line box.

<percentage>

Raise (positive value) or lower (negative value) the box by this distance (a percentage of the "line-height" value). The value "0%" means the same as "baseline".

<length>

Raise (positive value) or lower (negative value) the box by this distance. The value "0cm" means the same as "baseline".

NOTE: Values of this property have slightly different meanings in the context of tables. Please consult the section on table height algorithms for details.

XSL modifications to the CSS definition:

The CSS property shall be treated as a shorthand by XSL and maps as follows:

baseline

alignment-baseline="baseline"

alignment-adjust="auto"

baseline-shift="baseline"

dominant-baseline="auto"

top

alignment-baseline="top"

alignment-adjust="auto"

baseline-shift="baseline"

dominant-baseline="auto"

text-top

alignment-baseline="text-top"

alignment-adjust="auto"

baseline-shift="baseline"

dominant-baseline="auto"

middle

alignment-baseline="middle"

alignment-adjust="auto"

baseline-shift="baseline"

dominant-baseline="auto"

bottom

alignment-baseline="bottom"

alignment-adjust="auto"

baseline-shift="baseline"

dominant-baseline="auto"

text-bottom

alignment-baseline="text-bottom"

alignment-adjust="auto"

baseline-shift="baseline"

dominant-baseline="auto"

sub

alignment-baseline="baseline"

alignment-adjust="auto"

baseline-shift="sub"

dominant-baseline="auto"

super

alignment-baseline="baseline"

alignment-adjust="auto"

baseline-shift="super"

dominant-baseline="auto"

<percentage>

alignment-baseline="baseline"

alignment-adjust="<percentage>"

baseline-shift="baseline"

dominant-baseline="auto"

<length>

alignment-baseline="baseline"

alignment-adjust="<length>"

baseline-shift="baseline"

dominant-baseline="auto"

7.27.23. “white-space”

CSS2 Definition:

<i>Value:</i>	normal pre nowrap inherit
<i>Initial:</i>	normal
<i>Applies to:</i>	block-level elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

CSS2 Reference: <http://www.w3.org/TR/REC-CSS2/text.html#propdef-white-space>.

This property declares how whitespace inside the element is handled. Values have the following meanings:

normal

This value directs user agents to collapse sequences of whitespace, and break lines as necessary to fill line boxes. Additional line breaks may be created by occurrences of "\A" in generated content (e.g., for the BR element in HTML).

pre

This value prevents user agents from collapsing sequences of whitespace. Lines are only broken at newlines in the source, or at occurrences of "\A" in generated content.

nowrap

This value collapses whitespace as for 'normal', but suppresses line breaks within text except for those created by "\A" in generated content (e.g., for the BR element in HTML).

Conforming user agents may ignore the 'white-space' property in author and user style sheets but must specify a value for it in the default style sheet.

XSL modifications to the CSS definition:

XSL splits control of whitespace collapsing, space and linefeed handling, and wrapping into separate properties.

The CSS property shall be treated as a shorthand by XSL and maps as follows:

normal

```
linefeed-treatment="treat-as-space"
white-space-collapse="true"
space-treatment="preserve"
wrap-option="wrap"
```

pre

```
linefeed-treatment="preserve"
```


white-space-collapse="false"

space-treatment="preserve"

wrap-option="no-wrap"

nowrap

linefeed-treatment="treat-as-space"

white-space-collapse="true"

space-treatment="preserve"

wrap-option="no-wrap"

7.27.24. “xml:lang”

XSL Definition:

<i>Value:</i>	<country-language> inherit
<i>Initial:</i>	not defined for shorthand properties
<i>Applies to:</i>	see prose
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Values have the following meanings:

<string>

A language and/or country specifier in conformance with [\[RFC1766\]](#).

Specifies the language and country to be used by the formatter in linguistic services (such as hyphenation) and in the determination of line breaks. This affects line composition in a system-dependent way.

The string may be any RFC 1766 code.

XSL treats xml:lang as a shorthand and uses it to set the country and language properties.

NOTE: In general, linguistic services (line-justification strategy, line-breaking and hyphenation) may depend on a combination of the "language", "script", and "country" properties.

8. Conformance

This specification defines three levels of conformance, in order of completeness:

Basic

includes the set of formatting objects and properties needed to support a minimum level of pagination or aural rendering.

Extended

includes everything else, except for shorthands. It is intended for applications whose goal is to provide sophisticated pagination.

Complete

includes everything.

Conformance to this specification is expressed in terms of conformance to any of the above levels.

An application that claims conformance to a given level of this specification must implement all the formatting objects and properties that apply to it for a given medium.

[Appendix B – Formatting Object Summary](#) on page 349 specifies which formatting objects belong to each of the above levels, and for what medium.

[Appendix C – Property Summary](#) on page 352 specifies which properties belong to each of the above levels.

The minimum level of conformance is Basic. A minimally conformant implementation is required to process as specified all the formatting objects and properties defined for the Basic level of the implementation's target medium.

Implementations may choose to process formatting objects from levels or target media other than the one to which they conform. In order to ensure interoperability, this specification defines a fallback for each formatting object in the Extended and Complete levels.

An implementation cannot claim conformance to a given level if any of the formatting objects at that level is implemented as the fallback specified here for that level. Correct processing of fallbacks does not constitute conformance.

Conforming implementations must support at least one of the "writing-mode" values defined in this Recommendation. Although writing-mode is defined as a Basic property with an initial value of "lr-tb", it is not the intention of this specification to impose this particular, or any other, writing mode value on conformant applications. If an implementation does not support a writing-mode used in a stylesheet, either explicitly or by relying on the initial value, it should display either an "unsupported character glyph" message or display some indication that the content cannot be correctly rendered.

Appendix A. Internationalization

A.1. Additional “writing-mode” values

The following additional values for the "writing-mode" property provide for more extensive internationalization support.

The values have the following meanings:

tb-lr

Inline components and text within a line are stacked top-to-bottom. Lines and blocks are stacked left-to-right.

Establishes the following directions:

- inline-progression-direction to top-to-bottom
- block-progression-direction to left-to-right
- shift-direction to right-to-left

bt-lr

Inline components and text within a line are stacked bottom-to-top. Lines and blocks are stacked left-to-right.

Establishes the following directions:

- inline-progression-direction to bottom-to-top
- block-progression-direction to left-to-right
- shift-direction to right-to-left

bt-rl

Inline components and text within a line are stacked bottom-to-top. Lines and blocks are stacked right-to-left.

Establishes the following directions:

- inline-progression-direction to bottom-to-top
- block-progression-direction to right-to-left
- shift-direction to left-to-right

lr-bt

Inline components and text within a line are stacked left-to-right. Lines and blocks are stacked bottom-to-top.

Establishes the following directions:

- inline-progression-direction to left-to-right
- block-progression-direction to bottom-to-top
- shift-direction to bottom-to-top

rl-bt

Inline components and text within a line are stacked right-to-left. Lines and blocks are stacked bottom-to-top.

Establishes the following directions:

- inline-progression-direction to right-to-left
- block-progression-direction to bottom-to-top
- shift-direction to bottom-to-top

lr-alternating-rl-bt

Inline components and text within the first line are stacked left-to-right, within the second line

they are stacked right-to-left; continuing in alternation. Lines and blocks are stacked bottom-to-top.

Establishes the following directions:

- inline-progression-direction to left-to-right for odd-numbered lines, right-to-left for even-numbered lines
- block-progression-direction to bottom-to-top
- shift-direction to bottom-to-top

lr-alternating-rl-tb

Inline components and text within the first line are stacked left-to-right, within the second line they are stacked right-to-left; continuing in alternation. Lines and blocks are stacked top-to-bottom.

Establishes the following directions:

- inline-progression-direction to left-to-right for odd-numbered lines, right-to-left for even-numbered lines
- block-progression-direction to top-to-bottom
- shift-direction to bottom-to-top

lr-inverting-rl-bt

Inline components and text within the first line are stacked left-to-right, within the second line they are inverted and are stacked right-to-left; continuing in alternation. Lines and blocks are stacked bottom-to-top.

Establishes the following directions:

- inline-progression-direction to left-to-right for odd-numbered lines, right-to-left for even-numbered lines
- block-progression-direction to bottom-to-top
- shift-direction to bottom-to-top for odd-numbered lines, top-to-bottom for even-numbered lines

lr-inverting-rl-tb

Inline components and text within the first line are stacked left-to-right, within the second line they are inverted and are stacked right-to-left; continuing in alternation. Lines and blocks are stacked top-to-bottom.

Establishes the following directions:

- inline-progression-direction to left-to-right for odd-numbered lines, right-to-left for even-numbered lines
- block-progression-direction to top-to-bottom
- shift-direction to bottom-to-top for odd-numbered lines, top-to-bottom for even-numbered lines

tb-lr-in-lr-pairs

Text is written in two character, left-to-right, pairs. The pairs are then stacked top-to-bottom to form a line. Lines and blocks are stacked left-to-right.

Establishes the following directions:

- inline-progression-direction to top-to-bottom
- block-progression-direction to left-to-right
- shift-direction to right-to-left

The two glyph areas in the pair are aligned in the inline-progression-direction in the same manner as for lines with a left-to-right inline progression direction using the values of the "alignment-baseline", "alignment-adjust", "baseline-shift", and "dominant-baseline" properties.

NOTE: Informally: the two glyph areas are placed with respect to each other vertically as if they were part of a line written left-to-right.

For stacking into lines each pair is considered a glyph area with an allocation rectangle that is the minimum rectangle required to enclose the allocation rectangles of the glyphs in the pair.

In the block-progression-direction the pairs are aligned on a line that is half-way between the end-edge of the first glyph area and the start-edge of the second glyph area in the pair.

Appendix B. Formatting Object Summary

This section contains tables summarizing the conformance level of each of the defined formatting objects, i.e., basic or extended. For a description of basic and extended, see § 8 – Conformance on page 345. Included with each formatting object name is a designation of its inclusion or exclusion from the basic set of formatting objects for the particular class, e.g., aural. A proposed fallback treatment is also specified.

B.1. Declaration and Pagination and Layout Formatting Objects

Formatting Object	Visual	Aural
fo:root	basic	basic
fo:page-sequence	basic	basic
fo:page-sequence-master	basic	basic
fo:single-page-master-reference	basic	basic
fo:repeatable-page-master-reference	basic	basic
fo:repeatable-page-master-alternatives	extended fallback: use the page-master referenced in the first fo:conditional-page-master-reference child	extended fallback: use the page-master referenced in the first fo:conditional-page-master-reference child
fo:conditional-page-master-reference	extended fallback: use the page-master referenced in the first fo:conditional-page-master-reference child	extended fallback: use the page-master referenced in the first fo:conditional-page-master-reference child

fo:layout-master-set	basic	basic
fo:simple-page-master	basic	basic
fo:region-body	basic	basic
fo:region-before	extended fallback: include after content of body region is placed	extended fallback: include after content of body region is spoken
fo:region-after	extended fallback: include after content of body region is placed	extended fallback: include after content of body region is spoken
fo:region-start	extended fallback: include after content of body region is placed	extended fallback: include after content of body region is spoken
fo:region-end	extended fallback: include after content of body region is placed	extended fallback: include after content of body region is spoken
fo:declarations	basic	basic
fo:color-profile	extended fallback: ignore, use the sRGB fallback of the icc-color function	N/A
fo:flow	basic	basic
fo:static-content	extended fallback: include after content of body region is placed	extended fallback: include after content of body region is spoken
fo:title	extended fallback: include before content of body region is placed	extended fallback: include before content of body region is spoken

B.2. Block Formatting Objects

Formatting Object	Visual	Aural
fo:block	basic	basic
fo:block-container	extended fallback: display indication that content cannot be correctly rendered	basic

B.3. Inline Formatting Objects

Formatting Object	Visual	Aural
fo:bidi-override	extended fallback: display indication that content cannot be correctly rendered.	basic
fo:character	basic	basic
fo:initial-property-set	extended fallback: ignore any properties specified on this object.	basic
fo:external-graphic	basic	basic
fo:instream-foreign-object	extended fallback: display an indication that content cannot be correctly rendered.	extended fallback: speak an indication that content cannot be correctly spoken.

fo:inline-container	extended fallback: display indication that content cannot be correctly rendered.	extended fallback: speak an indication that content cannot be correctly spoken.
fo:leader	basic	basic
fo:page-number	basic	extended fallback: speak an indication that content cannot be correctly spoken.
fo:page-number-citation	extended fallback: display an indication that content cannot be correctly rendered.	extended fallback: speak an indication that content cannot be correctly spoken.

B.4. Table Formatting Objects

Formatting Object	Visual	Aural
fo:table-and-caption	basic	basic
fo:table	basic	basic
fo:table-column	basic	basic
fo:table-caption	extended fallback: <ul style="list-style-type: none"> caption-side="start" becomes caption-side="before" caption-side="end" becomes caption-side="after" caption-side="left" becomes caption-side="before" caption-side="right" becomes caption-side="after" 	extended fallback: <ul style="list-style-type: none"> caption-side="start" becomes caption-side="before" caption-side="end" becomes caption-side="after" caption-side="left" becomes caption-side="before" caption-side="right" becomes caption-side="after"
fo:table-header	basic	basic
fo:table-footer	extended fallback: place at end of table.	extended fallback: speak at end of table
fo:table-body	basic	basic
fo:table-row	basic	basic
fo:table-cell	basic	basic

B.5. List Formatting Objects

Formatting Object	Visual	Aural
fo:list-block	basic	basic
fo:list-item	basic	basic
fo:list-item-body	basic	basic
fo:list-item-label	extended fallback: labels that break across multiple lines are treated as separate blocks before list-item-body.	basic

B.6. Link and Multi Formatting Objects

Formatting Object	Visual	Aural
-------------------	--------	-------

fo:basic-link	extended fallback: promote content to parent formatting object.	extended fallback: promote content to parent formatting object..
fo:multi-switch	extended fallback: utilize the contents of the first eligible multi-case formatting object.	extended fallback: utilize the contents of the first eligible multi-case formatting object.
fo:multi-case	basic: needed as wrapper for fallback for multi-switch	basic: needed as wrapper for fallback for multi-switch
fo:multi-toggle	extended fallback: promote content to parent formatting object.	extended fallback: promote content to parent formatting object.
fo:multi-properties	extended fallback: promote content to parent formatting object.	extended fallback: promote content to parent formatting object.
fo:multi-property-set	extended fallback: ignore.	extended fallback: ignore.

B.7. Out-of-line Formatting Objects

Formatting Object	Visual	Aural
fo:float	extended fallback: place inline.	extended fallback: place inline.
fo:footnote	extended fallback: place inline.	extended fallback: place inline..
fo:footnote-body	extended fallback: place inline.	extended fallback: place inline.

B.8. Other Formatting Objects

Formatting Object	Visual	Aural
fo:wrapper	basic	basic
fo:marker	extended fallback: ignore.	extended fallback: ignore.
fo:retrieve-marker	extended fallback: display indication that content cannot be correctly rendered.	extended fallback: speak an indication that content cannot be correctly rendered.

Appendix C. Property Summary

C.1. Explanation of Trait Mapping Values

Rendering

Maps directly into a rendering trait of the same name.

Disappears

There is no trait mapping.

Shorthand

A shorthand that is mapped into one or more properties. There are no traits associated with a shorthand property. The traits are associated with the individual properties.

Refine

Disappears in Refinement. During refinement it sets up one or more other traits.

Formatting

Maps directly into a formatting trait of the same name.

Specification

Sub-class of formatting. It is the same as a formatting trait, but is specified on formatting objects that are referenced.

See prose

Used to calculate a formatting trait, which does not have the same name as the property. Other properties may also influence the trait value. See the property description for details.

Font selection

Property that participates in font selection.

Value change

Maps to a trait of the same name, but the value is not just copied.

Reference

An association between two names. Establishes a reference within the formatting object tree.

Action

Behavior trait.

Magic

Handled by the formatter in an implementation-defined way. There are no specific traits for this property.

C.2. Property Table: Part I

Name	Values	Initial Value	Inherited	Percentages
“absolute-position”	auto absolute fixed inherit	auto	no	N/A
“active-state”	link visited active hover focus	no, a value is required	no	N/A

Name	Values	Initial Value	Inherited	Percentages
“alignment-adjust”	auto baseline before-edge text-before-edge middle central after-edge text-after-edge ideographic alphabetic hanging mathematical <percentage> <length> inherit	auto	no	see prose
“alignment-baseline”	auto baseline before-edge text-before-edge middle central after-edge text-after-edge ideographic alphabetic hanging mathematical inherit	auto	no	N/A
“auto-restore”	yes no	no	yes	N/A
“azimuth”	<angle> [[left-side far-left left center-left center center-right right far-right right-side] behind] leftwards rightwards inherit	center	yes	N/A
“background”	[<background-color> <background-image> <background-repeat> <background-attachment> <background-position>] inherit	not defined for shorthand properties	no	allowed on 'background-position'
“background-attachment”	scroll fixed inherit	scroll	no	N/A
“background-color”	<color> transparent inherit	transparent	no	N/A
“background-image”	<uri-specification> none inherit	none	no	N/A
“background-position”	[[<percentage> <length>] { 1,2 } [[top center bottom] [left center right]]] inherit	0% 0%	no	refer to the size of the box itself
“background-position-horizontal”	<percentage> <length> left center right inherit	0%	no	refer to the size of the padding-rectangle
“background-position-vertical”	<percentage> <length> top center bottom inherit	0%	no	refer to the size of the padding-rectangle
“background-repeat”	repeat repeat-x repeat-y no-repeat inherit	repeat	no	N/A
“baseline-shift”	baseline sub super <percentage> <length> inherit	baseline	no	refers to the "line-height" of the parent area
“blank-or-not-blank”	blank not-blank any inherit	any	no	N/A
“block-progression-dimension”	auto <length> <percentage> <length-range> inherit	auto	no	see prose

Name	Values	Initial Value	Inherited	Percentages
“border”	[<border-width> <border-style> <color>] inherit	see individual properties	no	N/A
“border-after-color”	<color> inherit	the value of the 'color' property	no	N/A
“border-after-precedence”	force <integer> inherit	fo:table: 1, fo:table-cell: 0	no	N/A
“border-after-style”	<border-style> inherit	none	no	N/A
“border-after-width”	<border-width> <length-conditional> inherit	medium	no	N/A
“border-before-color”	<color> inherit	the value of the 'color' property	no	N/A
“border-before-precedence”	force <integer> inherit	fo:table: 1, fo:table-cell: 0	no	N/A
“border-before-style”	<border-style> inherit	none	no	N/A
“border-before-width”	<border-width> <length-conditional> inherit	medium	no	N/A
“border-bottom”	[<border-top-width> <border-style> <color>] inherit	see individual properties	no	N/A
“border-bottom-color”	<color> inherit	the value of the 'color' property	no	N/A
“border-bottom-style”	<border-style> inherit	none	no	N/A
“border-bottom-width”	<border-width> inherit	medium	no	N/A
“border-collapse”	collapse separate inherit	collapse	yes	N/A
“border-color”	<color>{ 1,4 } transparent inherit	see individual properties	no	N/A
“border-end-color”	<color> inherit	the value of the 'color' property	no	N/A
“border-end-precedence”	force <integer> inherit	fo:table: 1, fo:table-cell: 0	no	N/A
“border-end-style”	<border-style> inherit	none	no	N/A
“border-end-width”	<border-width> inherit	medium	no	N/A
“border-left”	[<border-top-width> <border-style> <color>] inherit	see individual properties	no	N/A
“border-left-color”	<color> inherit	the value of the 'color' property	no	N/A
“border-left-style”	<border-style> inherit	none	no	N/A
“border-left-width”	<border-width> inherit	medium	no	N/A
“border-right”	[<border-top-width> <border-style> <color>] inherit	see individual properties	no	N/A
“border-right-color”	<color> inherit	the value of the 'color' property	no	N/A

Name	Values	Initial Value	Inherited	Percentages
“border-right-style”	<border-style> inherit	none	no	N/A
“border-right-width”	<border-width> inherit	medium	no	N/A
“border-separation”	<length-bp-ip-direction> inherit	border-separation-direction="0pt" inline-separation-direction="0pt"	yes	N/A
“border-spacing”	<length> <length>? inherit	0pt	yes	N/A
“border-start-color”	<color> inherit	the value of the 'color' property	no	N/A
“border-start-precedence”	force <integer> inherit	fo:table: 1, fo:table-cell: 0	no	N/A
“border-start-style”	<border-style> inherit	none	no	N/A
“border-start-width”	<border-width> inherit	medium	no	N/A
“border-style”	<border-style>{ 1,4 } inherit	see individual properties	no	N/A
“border-top”	[<border-top-width> <border-style> <color>] inherit	see individual properties	no	N/A
“border-top-color”	<color> inherit	the value of the 'color' property	no	N/A
“border-top-style”	<border-style> inherit	none	no	N/A
“border-top-width”	<border-width> inherit	medium	no	N/A
“border-width”	<border-width>{ 1,4 } inherit	see individual properties	no	N/A
“bottom”	<length> <percentage> auto inherit	auto	no	refer to height of containing block
“break-after”	auto column page even-page odd-page inherit	auto	no	N/A
“break-before”	auto column page even-page odd-page inherit	auto	no	N/A
“caption-side”	before after start end top bottom left right inherit	before	yes	N/A
“case-name”	<name>	none, a value is required	no, a value is required	N/A
“case-title”	<string>	none, a value is required	no, a value is required	N/A
“character”	<character>	N/A, value is required	no, a value is required	N/A
“clear”	start end left right both none inherit	none	no	N/A
“clip”	<shape> auto inherit	auto	no	N/A
“color”	<color> inherit	depends on user agent	yes	N/A
“color-profile-name”	<name> inherit		no	N/A
“column-count”	<number> inherit	1	no	N/A

Name	Values	Initial Value	Inherited	Percentages
“column-gap”	<length> <percentage> inherit	12.0pt	no	refer to width of the region being divided into columns.
“column-number”	<number>	see prose	no	N/A
“column-width”	<length> <percentage>	see prose	no	refer to width of table
“content-height”	auto scale-to-fit <length> <percentage> inherit	auto	no	intrinsic height
“content-type”	<string> auto	auto	no	N/A
“content-width”	auto scale-to-fit <length> <percentage> inherit	auto	no	intrinsic height
“country”	none <country> inherit	none	yes	N/A
“cue”	<cue-before> <cue-after> inherit	not defined for shorthand properties	no	N/A
“cue-after”	<uri-specification> none inherit	none	no	N/A
“cue-before”	<uri-specification> none inherit	none	no	N/A
“destination-placement-offset”	<length>	0pt	no	N/A
“direction”	ltr rtl inherit	ltr	yes	N/A
“display-align”	auto before center after inherit	auto	yes	N/A
“dominant-baseline”	auto use-script no-change reset-size ideographic alphabetic hanging mathematical inherit	auto	no	N/A
“elevation”	<angle> below level above higher lower inherit	level	yes	N/A
“empty-cells”	show hide inherit	show	yes	N/A
“end-indent”	<length> inherit	0pt	yes	refer to width of containing reference-area
“ends-row”	yes no	no	no	N/A
“extent”	<length> <percentage> inherit	0.0pt	no	refer to the corresponding height or width of the region reference-area.
“external-destination”	<uri-specification>	null string	no	N/A
“float”	before start end left right none inherit	none	no	N/A
“flow-name”	<name>	an empty name	no, a value is required	N/A

Name	Values	Initial Value	Inherited	Percentages
“font”	[[<font-style> <font-variant> <font-weight>]? <font-size> [/ <line-height>]? <font-family>] caption icon menu message-box small-caption status-bar inherit	see individual properties	yes	N/A
“font-family”	[[<family-name> <generic-family>],]* [<family-name> <generic-family>] inherit	depends on user agent	yes	N/A
“font-height-override-after”	use-font-metrics <length> inherit	use-font-metrics	no	refer to font's em-height
“font-height-override-before”	use-font-metrics <length> inherit	use-font-metrics	no	refer to font's em-height
“font-selection-strategy”	auto character-by-character inherit	auto	yes	N/A
“font-size”	<absolute-size> <relative-size> <length> <percentage> inherit	medium	yes, the computed value is inherited	refer to parent element's font size
“font-size-adjust”	<number> none inherit	none	yes	N/A
“font-stretch”	normal wider narrower ultra-condensed extra-condensed condensed semi-condensed semi-expanded expanded extra-expanded ultra-expanded inherit	normal	yes	N/A
“font-style”	normal italic oblique backslant inherit	normal	yes	N/A
“font-variant”	normal small-caps inherit	normal	yes	N/A
“font-weight”	normal bold bolder lighter 100 200 300 400 500 600 700 800 900 inherit	normal	yes	N/A
“force-page-count”	auto even odd end-on-even end-on-odd no-force inherit	auto	no	N/A
“format”	<string>	1.	no	N/A
“glyph-orientation-horizontal”	<angle> inherit	0	yes	N/A
“glyph-orientation-vertical”	auto <angle> inherit	auto	yes	N/A
“grouping-separator”	<character>	no separator	no	N/A
“grouping-size”	<number>	no grouping	no	N/A
“height”	<length> <percentage> auto inherit	auto	no	see prose
“hyphenate”	false true inherit	false	yes	N/A
“hyphenation-character”	<character> inherit	The Unicode hyphen character u+2010	yes	N/A

Name	Values	Initial Value	Inherited	Percentages
“hyphenation-keep”	auto column page inherit	auto	yes	N/A
“hyphenation-ladder-count”	no-limit <number> inherit	no-limit	yes	N/A
“hyphenation-push-character-count”	<number> inherit	2	yes	N/A
“hyphenation-remain-character-count”	<number> inherit	2	yes	N/A
“id”	<id>	see prose	no, see prose	N/A
“indicate-destination”	yes no	no	no	N/A
“initial-page-number”	auto auto-odd auto-even <number> inherit	auto	no	N/A
“inline-progression-dimension”	auto <length> <percentage> <length-range> inherit	auto	no	see prose
“internal-destination”	null string <idref>	null string	no	N/A
“keep-together”	<keep> inherit	.within-line=auto, .within-column=auto, .within-page=auto	yes	N/A
“keep-with-next”	<keep> inherit	.within-line=auto, .within-column=auto, .within-page=auto	no	N/A
“keep-with-previous”	<keep> inherit	.within-line=auto, .within-column=auto, .within-page=auto	no	N/A
“language”	none <language> inherit	none	yes	N/A
“last-line-end-indent”	<length> <percentage> inherit	0pt	yes	width of containing block
“leader-alignment”	none reference-area page inherit	none	yes	N/A
“leader-length”	<length-range> inherit	leader-length-minimum=0pt, .optimum=12.0pt, .maximum=100%	yes	refer to width of content-rectangle of parent area
“leader-pattern”	space rule dots use-content inherit	space	yes	N/A
“leader-pattern-width”	use-font-metrics <length> inherit	use-font-metrics	yes	Refer to width of containing box
“left”	<length> <percentage> auto inherit	auto	no	refer to height of containing block
“letter-spacing”	normal <length> <space> inherit	normal	yes	N/A
“letter-value”	auto alphabetic traditional	auto	no	N/A
“linefeed-treatment”	ignore preserve treat-as-space inherit	treat-as-space	yes	N/A
“line-height”	normal <length> <number> <percentage> <space> inherit	normal	yes	refer to the font size of the element itself
“line-height-shift-adjustment”	consider-shifts disregard-shifts inherit	consider-shifts	yes	N/A
“line-stacking-strategy”	line-height font-height max-height inherit	line-height	yes	N/A

Name	Values	Initial Value	Inherited	Percentages
“margin”	<margin-width>{ 1,4} inherit	not defined for shorthand properties	no	refer to width of containing block
“margin-bottom”	<margin-width> inherit	0	no	refer to width of containing block
“margin-left”	<margin-width> inherit	Opt	no	refer to width of containing block
“margin-right”	<margin-width> inherit	Opt	no	refer to width of containing block
“margin-top”	<margin-width> inherit	0	no	refer to width of containing block
“marker-class-name”	<name>	an empty name	no, a value is required	N/A
“master-name”	<name>	an empty name	no, a value is required	N/A
“max-height”	<length> <percentage> none inherit	Opt	no	refer to height of containing block
“maximum-repeats”	<number> no-limit inherit	no-limit	no	N/A
“max-width”	<length> <percentage> none inherit	none	no	refer to width of containing block
“min-height”	<length> <percentage> inherit	Opt	no	refer to height of containing block
“min-width”	<length> <percentage> inherit	depends on UA	no	refer to width of containing block
“number-columns-repeated”	<number>	1	no	N/A
“number-columns-spanned”	<number>	1	no	N/A
“number-rows-spanned”	<number>	1	no	N/A
“odd-or-even”	odd even any inherit	any	no	N/A
“orphans”	<integer> inherit	2	yes	N/A
“overflow”	visible hidden scroll error-if-overflow auto inherit	auto	no	N/A
“padding”	<padding-width>{ 1,4} inherit	not defined for shorthand properties	no	refer to width of containing block
“padding-after”	<padding-width> <length-conditional> inherit	Opt	no	refer to width of containing block
“padding-before”	<padding-width> <length-conditional> inherit	Opt	no	refer to width of containing block
“padding-bottom”	<padding-width> inherit	Opt	no	refer to width of containing block
“padding-end”	<padding-width> inherit	Opt	no	refer to width of containing block
“padding-left”	<padding-width> inherit	Opt	no	refer to width of containing block
“padding-right”	<padding-width> inherit	Opt	no	refer to width of containing block

Name	Values	Initial Value	Inherited	Percentages
“padding-start”	<padding-width> inherit	0pt	no	refer to width of containing block
“padding-top”	<padding-width> inherit	0pt	no	refer to width of containing block
“page-break-after”	auto always avoid left right inherit	auto	no	N/A
“page-break-before”	auto always avoid left right inherit	auto	no	N/A
“page-break-inside”	avoid auto inherit	auto	yes	N/A
“page-height”	auto indefinite <length> inherit	auto	no	N/A
“page-position”	first last rest any inherit	any	no	N/A
“page-width”	auto indefinite <length> inherit	auto	no	N/A
“pause”	[<time> <percentage>]{1,2} inherit	depends on user agent	no	see descriptions of 'pause-before' and 'pause-after'
“pause-after”	<time> <percentage> inherit	depends on user agent	no	see prose
“pause-before”	<time> <percentage> inherit	depends on user agent	no	see prose
“pitch”	<frequency> x-low low medium high x-high inherit	medium	yes	N/A
“pitch-range”	<number> inherit	50	yes	N/A
“play-during”	<uri-specification> mix? repeat? auto none inherit	auto	no	N/A
“position”	static relative absolute fixed inherit	static	no	N/A
“precedence”	true false inherit	false	no	N/A
“provisional-distance-between-starts”	<length> inherit	24.0pt	yes	refer to width of the containing box
“provisional-label-separation”	<length> inherit	6.0pt	yes	refer to width of the containing box
“reference-orientation”	0 90 180 270 -90 -180 -270 inherit	0	yes (see prose)	N/A
“ref-id”	<idref> inherit	none, value required	no	N/A
“region-name”	xsl-region-body xsl-region-start xsl-region-end xsl-region-before xsl-region-after xsl-before-float-separator xsl-footnote-separator <name>	see prose	no, a value is required	N/A
“relative-align”	before baseline inherit	before	yes	N/A
“relative-position”	auto static relative inherit	static	no	N/A

Name	Values	Initial Value	Inherited	Percentages
“rendering-intent”	auto perceptual relative-colorimetric saturation absolute-colorimetric inherit	auto	no	N/A
“retrieve-boundary”	page page-sequence document	page-sequence	no	N/A
“retrieve-class-name”	<name>	an empty name	no, a value is required	N/A
“retrieve-position”	first-starting-within-page first-including-carryover last-starting-within-page last-ending-within-page	first-starting-within-page	no	N/A
“richness”	<number> inherit	50	yes	N/A
“right”	<length> <percentage> auto inherit	auto	no	refer to height of containing block
“role”	<string> <uri-specification> none inherit	none	no	N/A
“rule-style”	none dotted dashed solid double groove ridge inherit	solid	yes	N/A
“rule-thickness”	<length>	1.0pt	yes	N/A
“scaling”	uniform non-uniform inherit	uniform	no	N/A
“scaling-method”	auto integer-pixels resample-any-method inherit	auto	no	N/A
“score-spaces”	true false inherit	true	yes	N/A
“script”	none auto <script> inherit	auto	yes	N/A
“show-destination”	replace new	replace	no	N/A
“size”	<length>{ 1,2 } auto landscape portrait inherit	auto	N/A [XSL:no, is optional]	N/A
“source-document”	<uri-specification> [<uri-specification>]* none inherit	none	no	N/A
“space-after”	<space> inherit	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0	no	N/A (Differs from margin-bottom in CSS)
“space-before”	<space> inherit	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0	no	N/A (Differs from margin-top in CSS)

Name	Values	Initial Value	Inherited	Percentages
“space-end”	<space> inherit	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0	no	refer to the width of the containing area
“space-start”	<space> inherit	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0	no	refer to the width of the containing area
“space-treatment”	ignore preserve inherit	preserve	yes	N/A
“span”	none all inherit	none	no	N/A
“speak”	normal none spell-out inherit	normal	yes	N/A
“speak-header”	once always inherit	once	yes	N/A
“speak-numeral”	digits continuous inherit	continuous	yes	N/A
“speak-punctuation”	code none inherit	none	yes	N/A
“speech-rate”	<number> x-slow slow medium fast x-fast faster slower inherit	medium	yes	N/A
“src”	<uri-specification> inherit	none, value required	no	N/A
“start-indent”	<length> inherit	0pt	yes	refer to width of containing reference-area
“starting-state”	show hide	show	no	N/A
“starts-row”	yes no	no	no	N/A
“stress”	<number> inherit	50	yes	N/A
“suppress-at-line-break”	auto suppress retain inherit	auto	no	N/A
“switch-to”	xsl-preceding xsl-following xsl-any <name>[<name>]*	xsl-any	no	N/A
“table-layout”	auto fixed inherit	auto	no	N/A
“table-omit-footer-at-break”	yes no	no	no	N/A
“table-omit-header-at-break”	yes no	no	no	N/A
“target-presentation-context”	use-target-processing-context <uri-specification>	use-target-processing-context	no	N/A
“target-processing-context”	document-root <uri-specification>	document-root	no	N/A
“target-stylesheet”	use-normal-stylesheet <uri-specification>	use-normal-stylesheet	no	N/A
“text-align”	start center end justify inside outside left right <string> inherit	start	yes	N/A
“text-align-last”	relative start center end justify inside outside left right inherit	relative	yes	N/A

Name	Values	Initial Value	Inherited	Percentages
“text-decoration”	none [[underline no-underline] [overline no-overline] [line-through no-line-through] [blink no-blink]] inherit	none	no, but see prose	N/A
“text-indent”	<length> <percentage> inherit	Opt	yes	refer to width of containing block
“text-shadow”	none [<color> <length> <length> <length>? ,]* [<color> <length> <length> <length>?] inherit	none	no, see prose	N/A
“text-transform”	capitalize uppercase lowercase none	none	yes	N/A
“top”	<length> <percentage> auto inherit	auto	no	refer to height of containing block
“treat-as-word-space”	auto yes no inherit	auto	no	N/A
“unicode-bidi”	normal embed bidi-override inherit	normal	no	N/A
“vertical-align”	baseline middle sub super text-top text-bottom <percentage> <length> top bottom inherit	baseline	no	refer to the 'line-height' of the element itself
“visibility”	visible hidden collapse inherit	visible	no	N/A
“voice-family”	[[<specific-voice> <generic-voice>],]* [<specific-voice> <generic-voice>] inherit	depends on user agent	yes	N/A
“volume”	<number> <percentage> silent x-soft soft medium loud x-loud inherit	medium	yes	refer to inherited value
“white-space”	normal pre nowrap inherit	normal	yes	N/A
“white-space-collapse”	false true inherit	true	yes	N/A
“widows”	<integer> inherit	2	yes	N/A
“width”	<length> <percentage> auto inherit	auto	no	refer to width of containing block
“word-spacing”	normal <length> <space> inherit	normal	yes	N/A
“wrap-option”	no-wrap wrap inherit	wrap	yes	N/A
“writing-mode”	lr-tb rl-tb tb-rl lr rl tb inherit	lr-tb	yes (see prose)	N/A
“xml:lang”	<country-language> inherit	not defined for shorthand properties	yes	N/A
“z-index”	auto <integer> inherit	auto	no	N/A

C.3. Property Table: Part II

The Trait Mapping Values are explained in [Appendix C.1 – Explanation of Trait Mapping Values](#) on page 352.

Name	Values	Initial Value	Trait mapping	Core
“absolute-position”	auto absolute fixed inherit	auto	See prose.	Complete
“active-state”	link visited active hover focus	no, a value is required	Action	Extended. Fallback: N/A use fallback for fo:multi-properties
“alignment-adjust”	auto baseline before-edge text-before-edge middle central after-edge text-after-edge ideographic alphabetic hanging mathematical <percentage> <length> inherit	auto	Formatting	Basic
“alignment-baseline”	auto baseline before-edge text-before-edge middle central after-edge text-after-edge ideographic alphabetic hanging mathematical inherit	auto	Formatting	Basic
“auto-restore”	yes no	no	Action	Extended. Fallback: N/A use fallback for fo:multi-switch
“azimuth”	<angle> [[left-side far-left left center-left center center-right right far-right right-side] behind] leftwards rightwards inherit	center	Rendering	Basic
“background”	[<background-color> <background-image> <background-repeat> <background-attachment> <background-position>] inherit	not defined for shorthand properties	Shorthand	Complete
“background-attachment”	scroll fixed inherit	scroll	Rendering	Extended. Fallback: Initial value
“background-color”	<color> transparent inherit	transparent	Rendering	Basic
“background-image”	<uri-specification> none inherit	none	Rendering	Extended. Fallback: Initial value
“background-position”	[[<percentage> <length>]{1,2} [[top center bottom] [left center right]]] inherit	0% 0%	Shorthand	Complete

Name	Values	Initial Value	Trait mapping	Core
“background-position-horizontal”	<percentage> <length> left center right inherit	0%	Value change	Extended. Fallback: Initial value
“background-position-vertical”	<percentage> <length> top center bottom inherit	0%	Value change	Extended. Fallback: Initial value
“background-repeat”	repeat repeat-x repeat-y no-repeat inherit	repeat	Rendering	Extended. Fallback: no-repeat
“baseline-shift”	baseline sub super <percentage> <length> inherit	baseline	Formatting	Basic
“blank-or-not-blank”	blank not-blank any inherit	any	Specification	Extended. Fallback: N/A use fallback for for path generation
“block-progression-dimension”	auto <length> <percentage> <length-range> inherit	auto	Formatting	Basic
“border”	[<border-width> <border-style> <color>] inherit	see individual properties	Shorthand	Complete
“border-after-color”	<color> inherit	the value of the 'color' property	Rendering	Basic
“border-after-precedence”	force <integer> inherit	fo:table: 1, fo:table-cell: 0	Formatting	Basic
“border-after-style”	<border-style> inherit	none	Rendering	Basic
“border-after-width”	<border-width> <length-conditional> inherit	medium	Formatting and Rendering	Basic
“border-before-color”	<color> inherit	the value of the 'color' property	Rendering	Basic
“border-before-precedence”	force <integer> inherit	fo:table: 1, fo:table-cell: 0	Formatting	Basic
“border-before-style”	<border-style> inherit	none	Rendering	Basic
“border-before-width”	<border-width> <length-conditional> inherit	medium	Formatting and Rendering	Basic
“border-bottom”	[<border-top-width> <border-style> <color>] inherit	see individual properties	Shorthand	Complete
“border-bottom-color”	<color> inherit	the value of the 'color' property	Disappears	Basic
“border-bottom-style”	<border-style> inherit	none	Disappears	Basic
“border-bottom-width”	<border-width> inherit	medium	Disappears	Basic
“border-collapse”	collapse separate inherit	collapse	Formatting	Extended. Fallback: Initial value
“border-color”	<color>{1,4} transparent inherit	see individual properties	Shorthand	Complete

Name	Values	Initial Value	Trait mapping	Core
“border-end-color”	<color> inherit	the value of the 'color' property	Rendering	Basic
“border-end-precedence”	force <integer> inherit	fo:table: 1, fo:table-cell: 0	Formatting	Basic
“border-end-style”	<border-style> inherit	none	Rendering	Basic
“border-end-width”	<border-width> inherit	medium	Formatting and Rendering	Basic
“border-left”	[<border-top-width> <border-style> <color>] inherit	see individual properties	Shorthand	Complete
“border-left-color”	<color> inherit	the value of the 'color' property	Disappears	Basic
“border-left-style”	<border-style> inherit	none	Disappears	Basic
“border-left-width”	<border-width> inherit	medium	Disappears	Basic
“border-right”	[<border-top-width> <border-style> <color>] inherit	see individual properties	Shorthand	Complete
“border-right-color”	<color> inherit	the value of the 'color' property	Disappears	Basic
“border-right-style”	<border-style> inherit	none	Disappears	Basic
“border-right-width”	<border-width> inherit	medium	Disappears	Basic
“border-separation”	<length-bp-ip-direction> inherit	block-progression-direction="ltr" inline-progression-direction="ltr"	Formatting	Extended. Fallback: Initial value
“border-spacing”	<length> <length>? inherit	0pt	Shorthand	Complete
“border-start-color”	<color> inherit	the value of the 'color' property	Rendering	Basic
“border-start-precedence”	force <integer> inherit	fo:table: 1, fo:table-cell: 0	Formatting	Basic
“border-start-style”	<border-style> inherit	none	Rendering	Basic
“border-start-width”	<border-width> inherit	medium	Formatting and Rendering	Basic
“border-style”	<border-style>{ 1,4 } inherit	see individual properties	Shorthand	Complete
“border-top”	[<border-top-width> <border-style> <color>] inherit	see individual properties	Shorthand	Complete
“border-top-color”	<color> inherit	the value of the 'color' property	Disappears	Basic
“border-top-style”	<border-style> inherit	none	Disappears	Basic
“border-top-width”	<border-width> inherit	medium	Disappears	Basic
“border-width”	<border-width>{ 1,4 } inherit	see individual properties	Shorthand	Complete
“bottom”	<length> <percentage> auto inherit	auto	Formatting	Extended. Fallback: N/A due to fallback for absolute-position, relative-position

Name	Values	Initial Value	Trait mapping	Core
“break-after”	auto column page even-page odd-page inherit	auto	Formatting	Basic
“break-before”	auto column page even-page odd-page inherit	auto	Formatting	Basic
“caption-side”	before after start end top bottom left right inherit	before	Formatting	Complete
“case-name”	<name>	none, a value is required	Action	Extended. Fallback: N/A use fallback for fo:multi-switch
“case-title”	<string>	none, a value is required	Action	Extended. Fallback: N/A use fallback for fo:multi-switch
“character”	<character>	N/A, value is required	Formatting	Basic
“clear”	start end left right both none inherit	none	Formatting	Extended. Fallback: N/A use fallback for fo:float
“clip”	<shape> auto inherit	auto	Rendering	Basic
“color”	<color> inherit	depends on user agent	Rendering	Basic
“color-profile-name”	<name> inherit		Formatting	Extended. Fallback: N/A use fallback for fo:color-profile
“column-count”	<number> inherit	1	Specification	Extended. Fallback: Initial value
“column-gap”	<length> <percentage> inherit	12.0pt	Specification	Extended. Fallback: N/A due to fallback for column-count
“column-number”	<number>	see prose	Value change	Basic
“column-width”	<length> <percentage>	see prose	Specification	Basic
“content-height”	auto scale-to-fit <length> <percentage> inherit	auto	Formatting	Extended. Fallback: Initial value
“content-type”	<string> auto	auto	Formatting	Basic
“content-width”	auto scale-to-fit <length> <percentage> inherit	auto	Formatting	Extended. Fallback: Initial value
“country”	none <country> inherit	none	Formatting	Extended. Fallback: Initial value
“cue”	<cue-before> <cue-after> inherit	not defined for shorthand properties	Shorthand	Complete

Name	Values	Initial Value	Trait mapping	Core
“cue-after”	<uri-specification> none inherit	none	Rendering	Basic
“cue-before”	<uri-specification> none inherit	none	Rendering	Basic
“destination-placement-offset”	<length>	0pt	Action	Extended. Fallback: N/A use fallback for fo:basic-link
“direction”	ltr rtl inherit	ltr	See prose.	Basic
“display-align”	auto before center after inherit	auto	Formatting	Basic
“dominant-baseline”	auto use-script no-change reset-size ideographic alphabetic hanging mathematical inherit	auto	Formatting	Basic
“elevation”	<angle> below level above higher lower inherit	level	Rendering	Basic
“empty-cells”	show hide inherit	show	Formatting	Extended. Fallback: Initial value
“end-indent”	<length> inherit	0pt	Formatting	Basic
“ends-row”	yes no	no	Formatting	Extended. Fallback: Initial value
“extent”	<length> <percentage> inherit	0.0pt	Specification	Extended. Fallback: N/A use fallback for fo:region-before, fo:region-after, fo:region-start, and fo:region-end
“external-destination”	<uri-specification>	null string	Action	Extended. Fallback: N/A use fallback for fo:basic-link
“float”	before start end left right none inherit	none	Formatting	Extended. Fallback: N/A use fallback for fo:float
“flow-name”	<name>	an empty name	Reference	Basic
“font”	[[<font-style> <font-variant> <font-weight>]? <font-size> [/ <line-height>]? <font-family>] caption icon menu message-box small-caption status-bar inherit	see individual properties	Shorthand	Complete

Name	Values	Initial Value	Trait mapping	Core
“font-family”	[[<family-name> <generic-family>],]* [<family-name> <generic-family>] inherit	depends on user agent	Font selection	Basic
“font-height-override-after”	use-font-metrics <length> inherit	use-font-metrics	Formatting	Extended. Fallback: Initial value
“font-height-override-before”	use-font-metrics <length> inherit	use-font-metrics	Formatting	Extended. Fallback: Initial value
“font-selection-strategy”	auto character-by-character inherit	auto	Font selection	Complete
“font-size”	<absolute-size> <relative-size> <length> <percentage> inherit	medium	Formatting and Rendering	Basic
“font-size-adjust”	<number> none inherit	none	Font selection	Extended. Fallback: Initial value
“font-stretch”	normal wider narrower ultra-condensed extra-condensed condensed semi-condensed semi-expanded expanded extra-expanded ultra-expanded inherit	normal	Font selection	Extended. Fallback: Initial value
“font-style”	normal italic oblique backslant inherit	normal	Font selection	Basic
“font-variant”	normal small-caps inherit	normal	Font selection	Basic
“font-weight”	normal bold bolder lighter 100 200 300 400 500 600 700 800 900 inherit	normal	Font selection	Basic
“force-page-count”	auto even odd end-on-even end-on-odd no-force inherit	auto	Specification	Extended. Fallback: no-force
“format”	<string>	1.	Formatting	Basic
“glyph-orientation-horizontal”	<angle> inherit	0	Formatting	Extended. Fallback: Initial value
“glyph-orientation-vertical”	auto <angle> inherit	auto	Formatting	Extended. Fallback: Initial value
“grouping-separator”	<character>	no separator	Formatting	Basic
“grouping-size”	<number>	no grouping	Formatting	Basic
“height”	<length> <percentage> auto inherit	auto	Disappears	Basic
“hyphenate”	false true inherit	false	Formatting	Extended. Fallback: Initial value

Name	Values	Initial Value	Trait mapping	Core
“hyphenation-character”	<character> inherit	The Unicode hyphen character u+2010	Formatting	Extended. Fallback: N/A due to fallback for hyphenate
“hyphenation-keep”	auto column page inherit	auto	Formatting	Extended. Fallback: N/A due to fallback for hyphenate
“hyphenation-ladder-count”	no-limit <number> inherit	no-limit	Formatting	Extended. Fallback: N/A due to fallback for hyphenate
“hyphenation-push-character-count”	<number> inherit	2	Formatting	Extended. Fallback: N/A due to fallback for hyphenate
“hyphenation-remain-character-count”	<number> inherit	2	Formatting	Extended. Fallback: N/A due to fallback for hyphenate
“id”	<id>	see prose	Reference	Basic
“indicate-destination”	yes no	no	Action	Extended. Fallback: N/A use fallback for fo:basic-link
“initial-page-number”	auto auto-odd auto-even <number> inherit	auto	Formatting	Basic
“inline-progression-dimension”	auto <length> <percentage> <length-range> inherit	auto	Formatting	Basic
“internal-destination”	null string <idref>	null string	Action	Extended. Fallback: N/A use fallback for fo:basic-link
“keep-together”	<keep> inherit	.within-line=auto, .within-column=auto, .within-page=auto	Formatting	Extended. Fallback: Initial value
“keep-with-next”	<keep> inherit	.within-line=auto, .within-column=auto, .within-page=auto	Formatting	Basic
“keep-with-previous”	<keep> inherit	.within-line=auto, .within-column=auto, .within-page=auto	Formatting	Basic
“language”	none <language> inherit	none	Formatting	Extended. Fallback: Initial value
“last-line-end-indent”	<length> <percentage> inherit	0pt	Formatting	Extended. Fallback: Initial value

Name	Values	Initial Value	Trait mapping	Core
“leader-alignment”	none reference-area page inherit	none	Formatting	Extended. Fallback: Initial value
“leader-length”	<length-range> inherit	leader-length.minimum=0pt, .optimum=12.0pt, .maximum=100%	Formatting	Basic
“leader-pattern”	space rule dots use-content inherit	space	Formatting	Basic
“leader-pattern-width”	use-font-metrics <length> inherit	use-font-metrics	Formatting	Extended. Fallback: Initial value
“left”	<length> <percentage> auto inherit	auto	Formatting	Extended. Fallback: N/A due to fallback for absolute-position, relative-position
“letter-spacing”	normal <length> <space> inherit	normal	Disappears	Extended. Fallback: Initial value
“letter-value”	auto alphabetic traditional	auto	Formatting	Basic
“linefeed-treatment”	ignore preserve treat-as-space inherit	treat-as-space	Formatting	Extended. Fallback: Initial value
“line-height”	normal <length> <number> <percentage> <space> inherit	normal	Formatting	Basic
“line-height-shift-adjustment”	consider-shifts disregard-shifts inherit	consider-shifts	Formatting	Extended. Fallback: Initial value
“line-stacking-strategy”	line-height font-height max-height inherit	line-height	Formatting	Basic
“margin”	<margin-width>{ 1,4 } inherit	not defined for shorthand properties	Shorthand	Complete
“margin-bottom”	<margin-width> inherit	0	Disappears	Basic
“margin-left”	<margin-width> inherit	0pt	Disappears	Basic
“margin-right”	<margin-width> inherit	0pt	Disappears	Basic
“margin-top”	<margin-width> inherit	0	Disappears	Basic
“marker-class-name”	<name>	an empty name	Formatting	Extended. Fallback: N/A use fallback for fo:marker
“master-name”	<name>	an empty name	Specification	Basic
“max-height”	<length> <percentage> none inherit	0pt	Shorthand	Complete
“maximum-repeats”	<number> no-limit inherit	no-limit	Specification	Extended. Fallback: N/A use fallback for fo:repeat-maximum and fo:repeat-minimum

Name	Values	Initial Value	Trait mapping	Core
“max-width”	<length> <percentage> none inherit	none	Shorthand	Complete
“min-height”	<length> <percentage> inherit	0pt	Shorthand	Complete
“min-width”	<length> <percentage> inherit	depends on UA	Shorthand	Complete
“number-columns-repeated”	<number>	1	Specification	Basic
“number-columns-spanned”	<number>	1	Formatting	Basic
“number-rows-spanned”	<number>	1	Formatting	Basic
“odd-or-even”	odd even any inherit	any	Specification	Extended. Fallback: N/A use fallback for frequently used values
“orphans”	<integer> inherit	2	Formatting	Basic
“overflow”	visible hidden scroll error-if-overflow auto inherit	auto	Formatting	Basic
“padding”	<padding-width>{1,4} inherit	not defined for shorthand properties	Shorthand	Complete
“padding-after”	<padding-width> <length-conditional> inherit	0pt	Formatting and Rendering	Basic
“padding-before”	<padding-width> <length-conditional> inherit	0pt	Formatting and Rendering	Basic
“padding-bottom”	<padding-width> inherit	0pt	Disappears	Basic
“padding-end”	<padding-width> inherit	0pt	Formatting and Rendering	Basic
“padding-left”	<padding-width> inherit	0pt	Disappears	Basic
“padding-right”	<padding-width> inherit	0pt	Disappears	Basic
“padding-start”	<padding-width> inherit	0pt	Formatting and Rendering	Basic
“padding-top”	<padding-width> inherit	0pt	Disappears	Basic
“page-break-after”	auto always avoid left right inherit	auto	Shorthand	Complete
“page-break-before”	auto always avoid left right inherit	auto	Shorthand	Complete
“page-break-inside”	avoid auto inherit	auto	Shorthand	Complete
“page-height”	auto indefinite <length> inherit	auto	Specification	Basic
“page-position”	first last rest any inherit	any	Specification	Extended. Fallback: N/A use fallback for frequently used values
“page-width”	auto indefinite <length> inherit	auto	Specification	Basic
“pause”	[<time> <percentage>]{1,2} inherit	depends on user agent	Shorthand	Complete

Name	Values	Initial Value	Trait mapping	Core
“pause-after”	<time> <percentage> inherit	depends on user agent	Rendering	Basic
“pause-before”	<time> <percentage> inherit	depends on user agent	Rendering	Basic
“pitch”	<frequency> x-low low medium high x-high inherit	medium	Rendering	Basic
“pitch-range”	<number> inherit	50	Rendering	Basic
“play-during”	<uri-specification> mix? repeat? auto none inherit	auto	Rendering	Basic
“position”	static relative absolute fixed inherit	static	Shorthand	Complete
“precedence”	true false inherit	false	Specification	Extended. Fallback: N/A use fallback for fo:region-before, fo:region-after, fo:region-start, and fo:region-end
“provisional-distance-between-starts”	<length> inherit	24.0pt	Specification	Basic
“provisional-label-separation”	<length> inherit	6.0pt	Specification	Basic
“reference-orientation”	0 90 180 270 -90 -180 -270 inherit	0	See prose.	Basic
“ref-id”	<idref> inherit	none, value required	Reference	Extended. Fallback: N/A use fallback for fo:page-number-citation
“region-name”	xsl-region-body xsl-region-start xsl-region-end xsl-region-before xsl-region-after xsl-before-float-separator xsl-footnote-separator <name>	see prose	Specification	Basic
“relative-align”	before baseline inherit	before	Formatting	Basic
“relative-position”	auto static relative inherit	static	See prose.	Extended. Fallback: Initial value
“rendering-intent”	auto perceptual relative-colorimetric saturation absolute-colorimetric inherit	auto	Formatting	Extended. Fallback: N/A use fallback for fo:color-profile
“retrieve-boundary”	page page-sequence document	page-sequence	Formatting	Extended. Fallback: N/A use fallback for fo:retrieve-marker

Name	Values	Initial Value	Trait mapping	Core
“retrieve-class-name”	<name>	an empty name	Formatting	Extended. Fallback: N/A use fallback for fo:retrieve-marker
“retrieve-position”	first-starting-within-page first-including-carryover last-starting-within-page last-ending-within-page	first-starting-within-page	Formatting	Extended. Fallback: N/A use fallback for fo:retrieve-marker
“richness”	<number> inherit	50	Rendering	Basic
“right”	<length> <percentage> auto inherit	auto	Formatting	Extended. Fallback: N/A due to fallback for absolute-position, relative-position
“role”	<string> <uri-specification> none inherit	none	Rendering	Basic
“rule-style”	none dotted dashed solid double groove ridge inherit	solid	Rendering	Basic
“rule-thickness”	<length>	1.0pt	Rendering	Basic
“scaling”	uniform non-uniform inherit	uniform	Formatting	Extended. Fallback: Initial value
“scaling-method”	auto integer-pixels resample-any-method inherit	auto	Formatting	Extended. Fallback: Initial value
“score-spaces”	true false inherit	true	Formatting	Extended. Fallback: Initial value
“script”	none auto <script> inherit	auto	Formatting	Extended. Fallback: none
“show-destination”	replace new	replace	Action	Extended. Fallback: N/A use fallback for fo:basic-link
“size”	<length>{1,2} auto landscape portrait inherit	auto	Shorthand	Complete
“source-document”	<uri-specification> [<uri-specification>]* none inherit	none	Rendering	Basic
“space-after”	<space> inherit	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0	Formatting	Basic
“space-before”	<space> inherit	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0	Formatting	Basic

Name	Values	Initial Value	Trait mapping	Core
“space-end”	<space> inherit	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0	Formatting	Basic
“space-start”	<space> inherit	space.minimum=0pt, .optimum=0pt, .maximum=0pt, .conditionality=discard, .precedence=0	Formatting	Basic
“space-treatment”	ignore preserve inherit	preserve	Formatting	Extended. Fallback: Initial value
“span”	none all inherit	none	Formatting	Extended. Fallback: Initial value
“speak”	normal none spell-out inherit	normal	Rendering	Basic
“speak-header”	once always inherit	once	Rendering	Basic
“speak-numeral”	digits continuous inherit	continuous	Rendering	Basic
“speak-punctuation”	code none inherit	none	Rendering	Basic
“speech-rate”	<number> x-slow slow medium fast x-fast faster slower inherit	medium	Rendering	Basic
“src”	<uri-specification> inherit	none, value required	Reference	Basic
“start-indent”	<length> inherit	0pt	Formatting	Basic
“starting-state”	show hide	show	Action	Extended. Fallback: N/A use fallback for fo:multi-switch
“starts-row”	yes no	no	Formatting	Extended. Fallback: Initial value
“stress”	<number> inherit	50	Rendering	Basic
“suppress-at-line-break”	auto suppress retain inherit	auto	Formatting	Extended. Fallback: Initial value
“switch-to”	xsl-preceding xsl-following xsl-any <name>[<name>]*	xsl-any	Action	Extended. Fallback: N/A use fallback for fo:multi-switch
“table-layout”	auto fixed inherit	auto	Formatting	Extended. Fallback: fixed
“table-omit-footer-at-break”	yes no	no	Formatting	Extended. Fallback: Initial value
“table-omit-header-at-break”	yes no	no	Formatting	Extended. Fallback: Initial value

Name	Values	Initial Value	Trait mapping	Core
“target-presentation-context”	use-target-processing-context <uri-specification>	use-target-processing-context	Action	Extended. Fallback: N/A use fallback for fo:basic-link
“target-processing-context”	document-root <uri-specification>	document-root	Action	Extended. Fallback: N/A use fallback for fo:basic-link
“target-stylesheet”	use-normal-stylesheet <uri-specification>	use-normal-stylesheet	Action	Extended. Fallback: N/A use fallback for fo:basic-link
“text-align”	start center end justify inside outside left right <string> inherit	start	Value change	Basic
“text-align-last”	relative start center end justify inside outside left right inherit	relative	Value change	Extended. Fallback: Initial value
“text-decoration”	none [[underline no-underline] [overline no-overline] [line-through no-line-through] [blink no-blink]] inherit	none	See prose.	Extended. Fallback: Initial value
“text-indent”	<length> <percentage> inherit	0pt	Formatting	Basic
“text-shadow”	none [<color> <length> <length> <length>? ,]* [<color> <length> <length> <length>?] inherit	none	Rendering	Extended. Fallback: Initial value
“text-transform”	capitalize uppercase lowercase none	none	Refine	Extended. Fallback: Initial value
“top”	<length> <percentage> auto inherit	auto	Formatting	Extended. Fallback: N/A due to fallback for absolute-position, relative-position
“treat-as-word-space”	auto yes no inherit	auto	Formatting	Extended. Fallback: Initial value
“unicode-bidi”	normal embed bidi-override inherit	normal	Formatting	Extended. Fallback: See prose
“vertical-align”	baseline middle sub super text-top text-bottom <percentage> <length> top bottom inherit	baseline	Shorthand	Complete
“visibility”	visible hidden collapse inherit	visible	Magic	Basic

Name	Values	Initial Value	Trait mapping	Core
“voice-family”	[[<specific-voice> <generic-voice>],]* [<specific-voice> <generic-voice>] inherit	depends on user agent	Rendering	Basic
“volume”	<number> <percentage> silent x-soft soft medium loud x-loud inherit	medium	Rendering	Basic
“white-space”	normal pre nowrap inherit	normal	Shorthand	Complete
“white-space-collapse”	false true inherit	true	Formatting	Extended. Fallback: Initial value
“widows”	<integer> inherit	2	Formatting	Basic
“width”	<length> <percentage> auto inherit	auto	Disappears	Basic
“word-spacing”	normal <length> <space> inherit	normal	Disappears	Extended. Fallback: Initial value
“wrap-option”	no-wrap wrap inherit	wrap	Formatting	Basic
“writing-mode”	lr-tb rl-tb tb-rl lr rl tb inherit	lr-tb	See prose.	Basic
“xml:lang”	<country-language> inherit	not defined for shorthand properties	Shorthand	Complete
“z-index”	auto <integer> inherit	auto	Value change	Basic

Appendix D. References

D.1. Normative References

W3C XML

World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*. W3C Recommendation. See <http://www.w3.org/TR/1998/REC-xml-19980210>

W3C XML Names

World Wide Web Consortium. *Namespaces in XML*. W3C Recommendation. See <http://www.w3.org/TR/REC-xml-names>

XSLT

World Wide Web Consortium. *XSL Transformations (XSLT)*. W3C Recommendation. See <http://www.w3.org/TR/xslt>

XPath

World Wide Web Consortium. *XML Path Language*. W3C Recommendation. See <http://www.w3.org/TR/xpath>

RDF

World Wide Web Consortium. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. See <http://www.w3.org/TR/REC-rdf-syntax>

IEEE 754

Institute of Electrical and Electronics Engineers. *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985.

ISO31

International Organization for Standardization. *ISO 31:1992, Amended 1998. Quantities and units* International Standard.

ISO639

International Organization for Standardization. *ISO 639:1998. Code for the representation of names of languages* International Standard.

ISO639-2

International Organization for Standardization. *ISO 639-2:1998. Codes for the representation of names of languages — Part 2: Alpha-3 code*. International Standard.

ISO3166-1

International Organization for Standardization. *ISO 3166-1:1997. Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*. International Standard.

ISO3166-2

International Organization for Standardization. *ISO 3166-2:1998. Codes for the representation of names of countries and their subdivisions — Part 2: Country subdivision code*. International Standard.

ISO3166-3

International Organization for Standardization. *ISO 3166-3:1999. Codes for the representation of names of countries and their subdivisions — Part 3: Code for formerly used names of countries*. International Standard.

ISO15924

International Organization for Standardization. *ISO 15924:1998. Code for the representation of names of scripts*. Draft International Standard.

ISO10646

International Organization for Standardization, International Electrotechnical Commission. *ISO/IEC 10646:2000. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*. International Standard.

RFC1766

IETF. *RFC 1766. Tags for the Identification of Languages*. See <http://www.ietf.org/rfc/rfc1766.txt>.

RFC2070

IETF. *RFC 2070. Internationalization of the Hypertext Markup Language*. See <http://www.ietf.org/rfc/rfc2070.txt>.

RFC2396

IETF. *RFC 2396. Uniform Resource Identifiers (URI): Generic Syntax*. See <http://www.ietf.org/rfc/rfc2396.txt>.

UNICODE

Unicode Consortium. *The Unicode Standard, Version 3.0*. See <http://www.unicode.org/unicode/uni2book/u2.html>.

UNICODE TR9

Unicode Consortium. *Unicode Technical Report #9. Unicode 3.0 Bidirectional Algorithm*. Unicode Technical Report. See <http://www.unicode.org/unicode/reports/tr9/index.html>.

UNICODE Character Database

Unicode Consortium. The most up-to-date version of the files for the Unicode Character Database is found at <http://www.unicode.org/Public/UNIDATA/> or at <ftp://ftp.unicode.org/Public/UNIDATA/>.

ICC

International Color Consortium. *ICC Profile Format Specification, version 3.2, 1995*. See <ftp://sgigate.sgi.com/pub/icc/ICC32.pdf>.

D.2. Other References

CSS2

World Wide Web Consortium. *Cascading Style Sheets, level 2 (CSS2)*, as amended by Errata document 1999/11/04. W3C Recommendation. See <http://www.w3.org/TR/1998/REC-CSS2-19980512> and <http://www.w3.org/Style/css2-updates/REC-CSS2-19980512-errata.html>.

DSSSL

International Organization for Standardization, International Electrotechnical Commission. *ISO/IEC 10179:1996. Document Style Semantics and Specification Language (DSSSL)*. International Standard.

UNICODE TR10

Unicode Consortium. *Draft Unicode Technical Report #10. Unicode Collation Algorithm*. Draft Unicode Technical Report. See <http://www.unicode.org/unicode/reports/tr10/index.html>.

UNICODE TR20

Unicode Consortium. *DRAFT Unicode Technical Report #20. Unicode in XML and other Markup Languages* DRAFT Unicode Technical Report. See <http://www.unicode.org/unicode/reports/tr20/index.html>.

OpenType

Microsoft, Adobe. *OpenType specification v.1.2*. See <http://www.microsoft.com/truetype/tt/tt.htm>.

W3C XML Stylesheet

World Wide Web Consortium. *Associating stylesheets with XML documents*. W3C Working Draft. See <http://www.w3.org/TR/WD-xml-stylesheet>

JLS

J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. See <http://java.sun.com/docs/books/jls/index.html>.

Appendix E. Property Index

- “absolute-position” — § 7.4.1 on page 163
- “active-state” — § 7.20.1 on page 276
- “alignment-adjust” — § 7.11.1 on page 221
- “alignment-baseline” — § 7.11.2 on page 223
- “auto-restore” — § 7.20.2 on page 277
- “azimuth” — § 7.5.1 on page 166
- “background” — § 7.27.1 on page 324
- “background-attachment” — § 7.6.1 on page 172
- “background-color” — § 7.6.2 on page 173
- “background-image” — § 7.6.3 on page 173
- “background-position” — § 7.27.2 on page 324
- “background-position-horizontal” — § 7.6.5 on page 175
- “background-position-vertical” — § 7.6.6 on page 176
- “background-repeat” — § 7.6.4 on page 174
- “baseline-shift” — § 7.11.3 on page 225
- “blank-or-not-blank” — § 7.23.1 on page 288
- “block-progression-dimension” — § 7.12.1 on page 230
- “border” — § 7.27.3 on page 327
- “border-after-color” — § 7.6.10 on page 178
- “border-after-precedence” — § 7.24.1 on page 298
- “border-after-style” — § 7.6.11 on page 178
- “border-after-width” — § 7.6.12 on page 179

- “border-before-color” — § 7.6.7 on page 177
- “border-before-precedence” — § 7.24.2 on page 298
- “border-before-style” — § 7.6.8 on page 177
- “border-before-width” — § 7.6.9 on page 177
- “border-bottom” — § 7.27.4 on page 328
- “border-bottom-color” — § 7.6.22 on page 184
- “border-bottom-style” — § 7.6.23 on page 184
- “border-bottom-width” — § 7.6.24 on page 184
- “border-collapse” — § 7.24.3 on page 299
- “border-color” — § 7.27.5 on page 328
- “border-end-color” — § 7.6.16 on page 180
- “border-end-precedence” — § 7.24.4 on page 299
- “border-end-style” — § 7.6.17 on page 181
- “border-end-width” — § 7.6.18 on page 181
- “border-left” — § 7.27.6 on page 329
- “border-left-color” — § 7.6.25 on page 185
- “border-left-style” — § 7.6.26 on page 185
- “border-left-width” — § 7.6.27 on page 186
- “border-right” — § 7.27.7 on page 329
- “border-right-color” — § 7.6.28 on page 186
- “border-right-style” — § 7.6.29 on page 186
- “border-right-width” — § 7.6.30 on page 187
- “border-separation” — § 7.24.5 on page 300
- “border-spacing” — § 7.27.9 on page 330
- “border-start-color” — § 7.6.13 on page 179
- “border-start-precedence” — § 7.24.6 on page 300
- “border-start-style” — § 7.6.14 on page 180
- “border-start-width” — § 7.6.15 on page 180
- “border-style” — § 7.27.8 on page 329
- “border-top” — § 7.27.10 on page 330
- “border-top-color” — § 7.6.19 on page 181
- “border-top-style” — § 7.6.20 on page 182
- “border-top-width” — § 7.6.21 on page 183
- “border-width” — § 7.27.11 on page 331

-
- “bottom” — § 7.4.4 on page 165
 - “break-after” — § 7.17.1 on page 264
 - “break-before” — § 7.17.2 on page 265
 - “caption-side” — § 7.24.7 on page 300
 - “case-name” — § 7.20.3 on page 278
 - “case-title” — § 7.20.4 on page 278
 - “character” — § 7.14.1 on page 250
 - “clear” — § 7.16.1 on page 259
 - “clip” — § 7.18.1 on page 268
 - “color” — § 7.15.1 on page 258
 - “color-profile-name” — § 7.15.2 on page 258
 - “column-count” — § 7.23.2 on page 289
 - “column-gap” — § 7.23.3 on page 289
 - “column-number” — § 7.24.8 on page 302
 - “column-width” — § 7.24.9 on page 302
 - “content-height” — § 7.12.2 on page 231
 - “content-type” — § 7.26.1 on page 319
 - “content-width” — § 7.12.3 on page 232
 - “country” — § 7.8.1 on page 202
 - “cue” — § 7.27.12 on page 331
 - “cue-after” — § 7.5.2 on page 167
 - “cue-before” — § 7.5.3 on page 167
 - “destination-placement-offset” — § 7.20.5 on page 279
 - “direction” — § 7.25.1 on page 312
 - “display-align” — § 7.11.4 on page 226
 - “dominant-baseline” — § 7.11.5 on page 227
 - “elevation” — § 7.5.4 on page 167
 - “empty-cells” — § 7.24.10 on page 303
 - “end-indent” — § 7.9.8 on page 210
 - “ends-row” — § 7.24.11 on page 303
 - “extent” — § 7.23.4 on page 289
 - “external-destination” — § 7.20.6 on page 279
 - “float” — § 7.16.2 on page 262
 - “flow-name” — § 7.23.5 on page 290
-

-
- “font” — § 7.27.13 on page 332
 - “font-family” — § 7.7.2 on page 193
 - “font-height-override-after” — § 7.25.2 on page 313
 - “font-height-override-before” — § 7.25.3 on page 313
 - “font-selection-strategy” — § 7.7.3 on page 194
 - “font-size” — § 7.7.4 on page 195
 - “font-size-adjust” — § 7.7.6 on page 198
 - “font-stretch” — § 7.7.5 on page 197
 - “font-style” — § 7.7.7 on page 199
 - “font-variant” — § 7.7.8 on page 200
 - “font-weight” — § 7.7.9 on page 201
 - “force-page-count” — § 7.23.6 on page 290
 - “format” — § 7.22.1 on page 287
 - “glyph-orientation-horizontal” — § 7.25.4 on page 314
 - “glyph-orientation-vertical” — § 7.25.5 on page 315
 - “grouping-separator” — § 7.22.2 on page 287
 - “grouping-size” — § 7.22.3 on page 287
 - “height” — § 7.12.4 on page 233
 - “hyphenate” — § 7.8.4 on page 204
 - “hyphenation-character” — § 7.8.5 on page 204
 - “hyphenation-keep” — § 7.13.1 on page 240
 - “hyphenation-ladder-count” — § 7.13.2 on page 240
 - “hyphenation-push-character-count” — § 7.8.6 on page 205
 - “hyphenation-remain-character-count” — § 7.8.7 on page 205
 - “id” — § 7.26.2 on page 320
 - “indicate-destination” — § 7.20.7 on page 280
 - “initial-page-number” — § 7.23.7 on page 291
 - “inline-progression-dimension” — § 7.12.5 on page 234
 - “internal-destination” — § 7.20.8 on page 280
 - “keep-together” — § 7.17.3 on page 265
 - “keep-with-next” — § 7.17.4 on page 266
 - “keep-with-previous” — § 7.17.5 on page 267
 - “language” — § 7.8.2 on page 202
 - “last-line-end-indent” — § 7.13.3 on page 241
-

-
- “leader-alignment” — § 7.19.1 on page 273
 - “leader-length” — § 7.19.4 on page 275
 - “leader-pattern” — § 7.19.2 on page 273
 - “leader-pattern-width” — § 7.19.3 on page 274
 - “left” — § 7.4.5 on page 165
 - “letter-spacing” — § 7.14.2 on page 251
 - “letter-value” — § 7.22.4 on page 288
 - “linefeed-treatment” — § 7.13.7 on page 244
 - “line-height” — § 7.13.4 on page 241
 - “line-height-shift-adjustment” — § 7.13.5 on page 243
 - “line-stacking-strategy” — § 7.13.6 on page 243
 - “margin” — § 7.27.14 on page 333
 - “margin-bottom” — § 7.9.2 on page 206
 - “margin-left” — § 7.9.3 on page 207
 - “margin-right” — § 7.9.4 on page 208
 - “margin-top” — § 7.9.1 on page 206
 - “marker-class-name” — § 7.21.1 on page 284
 - “master-name” — § 7.23.8 on page 292
 - “max-height” — § 7.12.6 on page 235
 - “maximum-repeats” — § 7.23.9 on page 293
 - “max-width” — § 7.12.7 on page 236
 - “min-height” — § 7.12.8 on page 237
 - “min-width” — § 7.12.9 on page 237
 - “number-columns-repeated” — § 7.24.12 on page 304
 - “number-columns-spanned” — § 7.24.13 on page 304
 - “number-rows-spanned” — § 7.24.14 on page 305
 - “odd-or-even” — § 7.23.10 on page 294
 - “orphans” — § 7.17.6 on page 268
 - “overflow” — § 7.18.2 on page 269
 - “padding” — § 7.27.15 on page 334
 - “padding-after” — § 7.6.32 on page 188
 - “padding-before” — § 7.6.31 on page 187
 - “padding-bottom” — § 7.6.36 on page 189
 - “padding-end” — § 7.6.34 on page 189
-

- “padding-left” — § 7.6.37 on page 190
- “padding-right” — § 7.6.38 on page 190
- “padding-start” — § 7.6.33 on page 188
- “padding-top” — § 7.6.35 on page 189
- “page-break-after” — § 7.27.16 on page 334
- “page-break-before” — § 7.27.17 on page 335
- “page-break-inside” — § 7.27.18 on page 337
- “page-height” — § 7.23.11 on page 294
- “page-position” — § 7.23.12 on page 295
- “page-width” — § 7.23.13 on page 296
- “pause” — § 7.27.19 on page 338
- “pause-after” — § 7.5.5 on page 168
- “pause-before” — § 7.5.6 on page 168
- “pitch” — § 7.5.7 on page 168
- “pitch-range” — § 7.5.8 on page 169
- “play-during” — § 7.5.9 on page 169
- “position” — § 7.27.20 on page 338
- “precedence” — § 7.23.14 on page 296
- “provisional-distance-between-starts” — § 7.26.4 on page 321
- “provisional-label-separation” — § 7.26.3 on page 320
- “reference-orientation” — § 7.18.3 on page 270
- “ref-id” — § 7.26.5 on page 321
- “region-name” — § 7.23.15 on page 297
- “relative-align” — § 7.11.6 on page 229
- “relative-position” — § 7.18.4 on page 272
- “rendering-intent” — § 7.15.3 on page 258
- “retrieve-boundary” — § 7.21.4 on page 286
- “retrieve-class-name” — § 7.21.2 on page 285
- “retrieve-position” — § 7.21.3 on page 285
- “richness” — § 7.5.10 on page 169
- “right” — § 7.4.3 on page 165
- “role” — § 7.3.2 on page 162
- “rule-style” — § 7.19.5 on page 275
- “rule-thickness” — § 7.19.6 on page 276

-
- “[scaling](#)” — § 7.12.10 on page 238
 - “[scaling-method](#)” — § 7.12.11 on page 238
 - “[score-spaces](#)” — § 7.26.6 on page 321
 - “[script](#)” — § 7.8.3 on page 203
 - “[show-destination](#)” — § 7.20.9 on page 280
 - “[size](#)” — § 7.27.21 on page 339
 - “[source-document](#)” — § 7.3.1 on page 162
 - “[space-after](#)” — § 7.9.6 on page 209
 - “[space-before](#)” — § 7.9.5 on page 209
 - “[space-end](#)” — § 7.10.1 on page 211
 - “[space-start](#)” — § 7.10.2 on page 211
 - “[space-treatment](#)” — § 7.13.8 on page 245
 - “[span](#)” — § 7.18.5 on page 272
 - “[speak](#)” — § 7.5.11 on page 170
 - “[speak-header](#)” — § 7.5.12 on page 170
 - “[speak-numeral](#)” — § 7.5.13 on page 170
 - “[speak-punctuation](#)” — § 7.5.14 on page 171
 - “[speech-rate](#)” — § 7.5.15 on page 171
 - “[src](#)” — § 7.26.7 on page 322
 - “[start-indent](#)” — § 7.9.7 on page 210
 - “[starting-state](#)” — § 7.20.10 on page 281
 - “[starts-row](#)” — § 7.24.15 on page 305
 - “[stress](#)” — § 7.5.16 on page 171
 - “[suppress-at-line-break](#)” — § 7.14.3 on page 252
 - “[switch-to](#)” — § 7.20.11 on page 281
 - “[table-layout](#)” — § 7.24.16 on page 305
 - “[table-omit-footer-at-break](#)” — § 7.24.17 on page 306
 - “[table-omit-header-at-break](#)” — § 7.24.18 on page 306
 - “[target-presentation-context](#)” — § 7.20.12 on page 282
 - “[target-processing-context](#)” — § 7.20.13 on page 283
 - “[target-stylesheet](#)” — § 7.20.14 on page 284
 - “[text-align](#)” — § 7.13.9 on page 246
 - “[text-align-last](#)” — § 7.13.10 on page 247
 - “[text-decoration](#)” — § 7.14.4 on page 253
-

- “text-indent” — § 7.13.11 on page 248
- “text-shadow” — § 7.14.5 on page 254
- “text-transform” — § 7.14.6 on page 255
- “top” — § 7.4.2 on page 164
- “treat-as-word-space” — § 7.14.7 on page 256
- “unicode-bidi” — § 7.25.6 on page 316
- “vertical-align” — § 7.27.22 on page 341
- “visibility” — § 7.26.8 on page 322
- “voice-family” — § 7.5.17 on page 171
- “volume” — § 7.5.18 on page 172
- “white-space” — § 7.27.23 on page 344
- “white-space-collapse” — § 7.13.12 on page 249
- “widows” — § 7.17.7 on page 268
- “width” — § 7.12.12 on page 239
- “word-spacing” — § 7.14.8 on page 256
- “wrap-option” — § 7.13.13 on page 250
- “writing-mode” — § 7.25.7 on page 317
- “xml:lang” — § 7.27.24 on page 345
- “z-index” — § 7.26.9 on page 323

Appendix F. Acknowledgements (Non-Normative)

This specification was developed and approved for publication by the W3C XSL Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval. During the development of XSL the members of the XSL WG were:

Sharon Adler, IBM (Co-Chair); Anders Berglund, IBM; Perin Blanchard, Novell; Scott Boag, Lotus; Larry Cable, Sun; Jeff Caruso, Pageflex; James Clark; Peter Danielsen, Bell Labs; Don Day, IBM; Stephen Deach, Adobe; Dwayne Dicks, SoftQuad; Andrew Greene, Pageflex; Paul Grosso, Arbortext; Eduardo Gutentag, Sun; Juliane Harbarth, Software AG; Mickey Kimchi, Enigma; Chris Lilley, W3C; Chris Maden, Exemplary Technologies; Jonathan Marsh, Microsoft; Alex Milowski, Lexica; Boris Moore, RivCom; Steve Muench, Oracle; Scott Parnell, Xerox; Jeremy Richman, BroadVision; Vincent Quint, W3C; Dan Rapp, Novell; Gregg Reynolds, Datalogics; Jonathan Robie, Software AG; Mark Scardina, Oracle; Henry Thompson, University of Edinburgh; Philip Wadler, Bell Labs; Norman Walsh, Arbortext; Sanjiva Weerawarana, IBM; Steve Zilles, Adobe (Co-Chair)

The XSL Working Group wishes to thank RenderX, Inc. for producing the PDF version of this specification.

This document has been produced from the XML source through an intermediate XSL Formatting Object representation, converted to PDF using XEP - an XSL FO rendering engine by RenderX, Inc.

(<http://www.renderX.com>).

This page is intentionally left blank.