



# DocBook

## ***xsITNG***

# Reference

Norman Walsh

Version 2.1.0  
Updated: 07 Apr 2023

Copyright © 2020–2023 Norman Walsh.

## **Revision History**

---

- 2.1.0, 07 Apr 2023

This is the first official release of the “2.x” versions of the stylesheets. Version 2.1.0 introduces a number of new features and corrects major issues with localization and media object rendering, but is not entirely backwards compatible with the 1.x versions. The incompatibilities should mostly effect customization layers, not formatted documents, though the HTML output will be a little different in most cases. For a summary of the changes, see *Appendix A, Changes in version 2.1.0*.



## Table of Contents

Part I. Introduction . . . . .	13
Preface . . . . .	15
1. Overview . . . . .	17
1.1. What do the stylesheets do? . . . . .	17
1.2. Getting the stylesheets . . . . .	17
2. Using the stylesheets . . . . .	19
2.1. Using the Jar . . . . .	20
2.2. Using the Python script . . . . .	22
2.3. Run with Java . . . . .	24
2.4. Run with Docker . . . . .	24
2.5. Extension functions . . . . .	25
2.6. “Chunked” output . . . . .	27
2.7. Effectivity attributes and profiling . . . . .	30
2.8. Syntax highlighting . . . . .	33
2.9. Persistent table of contents . . . . .	33
2.10. On-page table of contents . . . . .	36
2.11. Print output (dead tree editions) . . . . .	38
2.12. EPUB output . . . . .	38
3. Customizing the stylesheets . . . . .	41
3.1. Changing stylesheet parameters . . . . .	41
3.2. Creating a customization layer . . . . .	42
3.3. Managing CSS stylesheets . . . . .	46
3.4. Managing media . . . . .	47
3.5. Controlling numeration . . . . .	51
3.6. Creating something completely different . . . . .	54
4. Localization . . . . .	59
4.1. Background . . . . .	59
4.2. Overview . . . . .	60
4.3. Localization files . . . . .	64
4.4. Customizing a localization . . . . .	69
4.5. Caveats . . . . .	72
5. Implementation details . . . . .	75
5.1. Customizing chunking . . . . .	75
5.2. Lengths and units . . . . .	75
5.3. Verbatim styles . . . . .	76
5.4. Processing mediaobjects . . . . .	78
5.5. Templates . . . . .	81
5.6. Annotations . . . . .	82
5.7. The pre- and post-processing pipeline . . . . .	84
6. Building the stylesheets . . . . .	87
6.1. Prerequisites . . . . .	88
6.2. Repository structure . . . . .	88
6.3. Build tasks . . . . .	90
6.4. Testing tasks . . . . .	91
6.5. Running XSpec . . . . .	93
A. Changes in version 2.1.0 . . . . .	95

## Table of Contents

Glossary.....	99
References.....	101
Acknowledgements.....	103
Index.....	105
Index by module.....	107
Part II. Reference.....	109
I. Parameter reference.....	111
\$additional-languages.....	112
\$align-char-default, \$align-char-pad, \$align-char-width.....	113
\$allow-eval.....	117
\$annotate-toc.....	118
\$annotation-collection.....	119
\$annotation-mark.....	120
\$annotation-placement.....	121
\$annotation-style.....	122
\$annotations-js.....	123
\$auto-toc.....	124
\$bibliography-collection.....	125
\$bibliography-style.....	126
\$books-inherit-from.....	127
\$books-number-from.....	128
\$callout-default-column.....	129
\$callout-unicode-start.....	130
\$chunk.....	131
\$chunk-exclude.....	132
\$chunk-include.....	133
\$chunk-nav.....	135
\$chunk-nav-js.....	136
\$chunk-output-base-uri.....	137
\$chunk-renumber-footnotes.....	139
\$chunk-section-depth.....	140
\$classsynopsis-indent.....	141
\$component-numbers.....	142
\$component_numbers_inherit.....	143
\$components-inherit-from.....	144
\$components-number-from.....	145
\$control-js.....	146
\$copyright-collapse-years.....	147
\$copyright-year-range-separator.....	148
\$copyright-year-separator.....	149
\$date-date-format, \$date-dateTime-format.....	150
\$dc-metadata.....	151
\$debug.....	152
\$default-float-style.....	156
\$default-language.....	157
\$default-length-magnitude, \$default-length-unit.....	158
\$default-theme.....	159
\$division-numbers.....	160

\$division-numbers-inherit.	161
\$divisions-inherit-from.	162
\$divisions-number-from.	163
\$docbook-transclusion.	164
\$dynamic-profile-error.	165
\$dynamic-profile-variables.	166
\$dynamic-profiles.	167
\$experimental-pmuj.	168
\$fallback-js.	169
\$footnote-numeration.	170
\$formal-object-title-placement.	171
\$formal-objects-inherit-from.	172
\$formal-objects-number-from.	173
\$formalgroup-nested-object-title-placement.	174
\$funcsynopsis-default-style.	175
\$funcsynopsis-table-threshold.	176
\$funcsynopsis-trailing-punctuation.	177
\$generate-html-page.	178
\$generate-index.	179
\$generate-nested-toe.	180
\$generate-toe.	181
\$generate-trivial-toc.	182
\$generated-id-root.	183
\$generated-id-sep.	184
\$generator-metadata.	185
\$gentext-language.	186
\$glossary-collection.	187
\$glossary-sort-entries.	188
\$html-extension.	189
\$image-ignore-scaling.	190
\$image-nominal-height.	191
\$image-nominal-width.	192
\$image-property-warning.	193
\$index-on-role, \$index-on-type.	194
\$index-show-entries.	195
\$indexed-section-groups.	196
\$lists-of-equations.	197
\$lists-of-examples.	198
\$lists-of-figures.	199
\$lists-of-procedures.	200
\$lists-of-tables.	201
\$local-conventions.	202
\$mathml-js.	204
\$mediaobject-accessibility.	205
\$mediaobject-details-placement.	208
\$mediaobject-exclude-extensions.	209
\$mediaobject-grouped-by-type.	210
\$mediaobject-input-base-uri.	211

## Table of Contents

\$mediaobject-output-base-uri.....	212
\$mediaobject-output-paths.....	213
\$mediaobject-video-element.....	214
\$message-level.....	215
\$nominal-page-width.....	216
\$number-single-appendix.....	217
\$olink-databases.....	218
\$orderedlist-item-numeration.....	219
\$othername-in-middle.....	220
\$output-media.....	221
\$oxy-markup.....	222
\$page-style.....	223
\$pagetoc-dynamic.....	224
\$pagetoc-elements.....	225
\$pagetoc-js.....	226
\$paper-size.....	227
\$persistent-toc.....	228
\$persistent-toc-css.....	229
\$persistent-toc-filename.....	230
\$persistent-toc-js.....	231
\$persistent-toc-search.....	232
\$personal-name-style.....	233
\$pixels-per-inch.....	234
\$procedure-step-numeration.....	235
\$productionset-lhs-rhs-separator.....	236
\$profile-arch.....	237
\$profile-audience.....	238
\$profile-condition.....	239
\$profile-conformance.....	240
\$profile-lang.....	241
\$profile-os.....	242
\$profile-outputformat.....	243
\$profile-revision.....	244
\$profile-revisionflag.....	245
\$profile-role.....	246
\$profile-security.....	247
\$profile-separator.....	248
\$profile-userlevel.....	249
\$profile-vendor.....	250
\$profile-wordsize.....	251
\$qandadiv-default-toc.....	252
\$qandaset-default-label.....	253
\$qandaset-default-toc.....	254
\$refentry-generate-name.....	255
\$refentry-generate-title.....	256
\$relax-ng-grammar.....	257
\$resource-base-uri.....	258
\$revhistory-style.....	259

\$section-numbers.....	262
\$section-numbers-inherit.....	263
\$section-toc-depth.....	264
\$sections-inherit-from.....	265
\$sections-number-from.....	266
\$segmentedlist-style.....	267
\$sets-inherit-from.....	270
\$sets-number-from.....	271
\$show-remarks.....	272
\$sidebar-as-aside.....	273
\$sort-collation.....	274
\$table-accessibility.....	275
\$table-footnote-numeration.....	278
\$theme-picker.....	279
\$transclusion-id-fixup.....	280
\$transclusion-link-scope.....	281
\$transclusion-prefix-separator.....	282
\$transclusion-suffix.....	283
\$transform-after.....	284
\$transform-before.....	285
\$transform-original.....	286
\$unwrap-paragraphs.....	287
\$use-docbook-css.....	288
\$use-minified-css.....	289
\$user-css-links.....	290
\$variablelist-term-length-threshold.....	291
\$verbatim-callouts.....	292
\$verbatim-line-style.....	294
\$verbatim-number-every-nth.....	295
\$verbatim-number-first-line.....	296
\$verbatim-number-minlines.....	297
\$verbatim-number-separator.....	298
\$verbatim-numbered-elements.....	299
\$verbatim-plain-style.....	300
\$verbatim-space.....	301
\$verbatim-style-default.....	302
\$verbatim-syntax-highlight-css.....	303
\$verbatim-syntax-highlight-languages.....	304
\$verbatim-syntax-highlighter.....	305
\$verbatim-table-style.....	307
\$verbatim-trim-trailing-blank-lines.....	308
\$warn-about-missing-localizations.....	309
\$xlink-arclist-after.....	310
\$xlink-arclist-before.....	311
\$xlink-arclist-sep.....	312
\$xlink-arclist-titlesep.....	313
\$xlink-icon-closed.....	314
\$xlink-icon-open.....	315

## Table of Contents

\$xlink-js.....	316
\$xlink-style.....	317
\$xlink-style-default.....	318
\$xspec.....	319
II. Variables reference.....	321
\$arg-choice-def-close-str, .....	322
\$err:DYNAMIC-PROFILE-EVAL-ERROR.....	326
\$err:DYNAMIC-PROFILE-SYNTAX-ERROR.....	327
\$err:INTERNAL-HIGHLIGHT-ERROR.....	328
\$err:INTERNAL-RENUMBER-ERROR.....	329
\$err:INVALID-AREAREFS.....	330
\$err:INVALID-CALS.....	331
\$err:INVALID-CONSTRAINT.....	332
\$err:INVALID-DYNAMIC-PROFILE-ERROR.....	333
\$err:INVALID-INJECT.....	334
\$err:INVALID-NAME-STYLE.....	335
\$err:INVALID-PRODUCTIONRECAP.....	336
\$err:INVALID-RESULTS-REQUESTED.....	337
\$err:INVALID-TEMPLATE.....	338
\$err:INVALID-TRANSFORM.....	339
\$v:personal-name-styles.....	340
\$v:VERSION.....	341
\$v:VERSION-ID.....	342
\$v:admonition-icons.....	343
\$v:annotation-close.....	344
\$v:as-json.....	345
\$v:as-xml.....	346
\$v:bridgehead-map.....	347
\$v:chunk.....	348
\$v:chunk-filter-namespaces.....	349
\$v:chunk-renumber-footnotes.....	350
\$v:custom-localizations.....	351
\$v:debug.....	352
\$v:formal-object-title-placement.....	353
\$v:formalgroup-nested-object-title-placement.....	354
\$v:highlight-js-head-elements.....	355
\$v:image-nominal-height.....	356
\$v:image-nominal-width.....	357
\$v:invisible-characters.....	358
\$v:localization-base-uri.....	359
\$v:media-type-default.....	360
\$v:media-type-map.....	361
\$v:mediaobject-details-placement.....	364
\$v:mediaobject-exclude-extensions.....	365
\$v:mediaobject input base-uri.....	366
\$v:mediaobject-output-base-uri.....	367
\$v:nominal-page-width.....	368
\$v:olink-databases.....	369

\$v:prism-js-head-elements.....	370
\$v:standard-transforms.....	371
\$v:templates.....	374
\$v:theme-list.....	375
\$v:title-groups.....	376
\$v:title_properties.....	378
\$v:titlepage-default.....	379
\$v:toc-close.....	380
\$v:toc-open.....	381
\$v:unit-scale.....	382
\$v:user-title-groups.....	384
\$v:user_title_properties.....	385
\$v:user-xref-groups.....	386
\$v:user_xref_properties.....	387
\$v:verbatim-callouts.....	388
\$v:verbatim-line-style.....	389
\$v:verbatim-number-every-nth.....	390
\$v:verbatim-number-first-line.....	391
\$v:verbatim-number-minlines.....	392
\$v:verbatim-numbered-elements.....	393
\$v:verbatim-plain-style.....	394
\$v:verbatim-properties.....	395
\$v:verbatim-space.....	396
\$v:verbatim-syntax-highlight-languages.....	397
\$v:verbatim-syntax-highlight-options.....	398
\$v:verbatim-syntax-highlight-pygments-options.....	400
\$v:verbatim-table-style.....	401
\$v:xref-groups.....	402
\$v:xref_properties.....	403
III. Function reference.....	405
ext:cwd.....	406
ext:image-metadata.....	407
ext:image-properties.....	409
ext:pygmentize.....	410
ext:pygmentize-available.....	411
ext:validate-with-relax-ng.....	412
ext:xinclude.....	414
f:absolute-length.....	415
f:attributes.....	416
f:cals-colsep.....	417
f:cals-rowsep.....	418
f:check_gentext.....	419
f:chunk.....	420
f:chunk-filename.....	421
f:chunk-title.....	422
f:css-length.....	423
f:css-property.....	424
f:date-format.....	425

## Table of Contents

f:empty-length.....	427
f:equal-lengths.....	428
f:generate-id.....	430
f:gentext.....	431
f:gentext-letters.....	432
f:gentext-letters-for-language.....	433
f:highlight-verbatim.....	434
f:href.....	435
f:id.....	436
f:in-scope-language.....	437
f:intra-number-separator.....	438
f:is-empty-length.....	439
f:is-true.....	440
f:l10n-language.....	441
f:l10n-token.....	442
f:label-separator.....	443
f:language.....	444
f:languages.....	445
f:length-string.....	446
f:length-units.....	447
f:locales.....	448
f:make-length.....	449
f:mediaobject-amend-uri.....	450
f:mediaobject-input-base-uri.....	451
f:mediaobject-type.....	452
f:mediaobject-viewport.....	453
f:number-separator.....	454
f:object-align.....	455
f:object-contentheight.....	456
f:object-contentwidth.....	457
f:object-height.....	458
f:object-properties.....	459
f:object-scale.....	463
f:object-scalefit.....	464
f:object-valign.....	465
f:object-width.....	466
f:orderedlist-item-number.....	467
f:orderedlist-item-numeration.....	468
f:orderedlist-startingnumber.....	469
f:parse-length.....	470
f:pi.....	471
f:pi-attributes.....	472
f:post-label-punctuation.....	473
f:refsection.....	474
f:relative-length.....	475
f:relative-path.....	476
f:section.....	477
f:section-depth.....	478

f:spaces.....	479
f:step-number.....	480
f:step-numeration.....	481
f:syntax-highlight.....	482
f:target.....	483
f:template.....	484
f:tokenize-on-char.....	485
f:unique-id.....	486
f:uri-scheme.....	487
f:verbatim-highlight.....	488
f:verbatim-numbered.....	489
f:verbatim-style.....	490
f:verbatim-trim-trailing.....	491
f:xlink-style.....	492
f:xpointer-idref.....	493
IV. Template reference.....	495
t:audio-fallback.....	496
t:biblioentry.....	497
t:bottom-nav.....	498
t:chunk-cleanup.....	499
t:chunk-footnotes.....	500
t:chunk-output.....	501
t:docbook.....	502
t:generate-index.....	503
t:index-zone-reference.....	504
t:inline.....	505
t:mediaobject-img.....	507
t:person-name.....	508
t:person-name-family-given.....	511
t:person-name-first-last.....	512
t:person-name-last-first.....	514
t:person-name-list.....	515
t:table-footnotes.....	516
t:top-nav.....	517
t:video-fallback.....	518
t:xlink.....	519
V. Mode reference.....	521
m:annotation-content.....	522
m:ansi.....	523
m:ansi-table.....	524
m:attributes.....	525
m:biblio690.....	526
m:biblioentry.....	527
m:bibliomixed.....	528
m:callout-bug.....	529
m:callout-link.....	530
m:chunk-cleanup.....	531
m:chunk-filename.....	532

## Table of Contents

m:chunk-output .....	533
m:chunk-title .....	534
m:chunk-write .....	535
m:copyright-years .....	536
m:crossref .....	537
<del>m:crossref-inherit-separator</del> .....	538
m:crossref-label .....	539
<del>m:crossref-label-separator</del> .....	540
m:crossref-number .....	541
m:crossref-number-separator .....	542
<del>m:crossref-prefix</del> .....	543
m:crossref-suffix .....	544
m:crossref-title .....	545
m:details .....	546
<del>m:details-attribute</del> .....	547
m:docbook .....	548
m:footnote-number .....	549
m:footnotes .....	550
m:generate-titlepage .....	551
m:gentext .....	552
m:gentext-list .....	553
m:headline .....	555
m:headline-label .....	556
<del>m:headline-label-separator</del> .....	557
m:headline-number .....	558
<del>m:headline-number-separator</del> .....	559
<del>m:headline-prefix</del> .....	560
<del>m:headline-suffix</del> .....	561
m:headline-title .....	562
m:highlight-options .....	563
m:html-body-script .....	564
m:html-head .....	565
m:html-head-last .....	566
m:html-head-links .....	567
m:html-head-script .....	568
m:htmltable .....	569
m:imagemap .....	570
m:index-div .....	571
m:index-primary .....	572
m:index-secondary .....	573
m:index-see .....	574
m:index-seealso .....	575
m:index-tertiary .....	576
m:kr .....	577
m:kr-args .....	578
m:kr-table .....	579
m:kr-table-args .....	580
m:link .....	581

m:list-of-equations.....	582
m:list-of-examples.....	583
m:list-of-figures.....	584
m:list-of-procedures.....	585
m:list-of-tables.....	586
m:list-of-titles.....	587
m:mediaobject-end.....	588
m:mediaobject-info.....	589
m:mediaobject-output-adjust.....	590
m:mediaobject-start.....	592
m:mediaobject-uris.....	593
m:persistent-toc.....	594
m:production-number.....	595
m:pygments-options.....	596
m:revhistory-list.....	597
m:revhistory-table.....	598
m:seglist-table.....	599
m:segtitle-in-seg.....	600
m:synopfragment-bug.....	601
m:synopsis.....	602
m:title.....	603
m:titlepage.....	604
m:to-uppercase.....	605
m:toc.....	606
m:toc-entry.....	607
m:toc-nested.....	608
VI. Processing instruction reference.....	609
DocBook-xslTNG-version.....	610
current-dateTime.....	611
db.....	612
system-property.....	613
B. GNU Free Documentation License.....	615

## List of Figures

2.1. The sample document: article.xml.....	21
2.2. Opening the ToC.....	34
2.3. The Persistent ToC.....	34
4.1. Sample book source.....	61
4.2. Sample book (annotated).....	62
4.3. Sample book source (French).....	63
4.4. Sample book (French).....	64
4.5. Example locale file (excerpted).....	65
4.6. Sample book (Alternate).....	71
1. An example of revhistory rendered as a table.....	261
2. An example of revhistory rendered as a list.....	261
1. Segmented list formatted as a table.....	268
2. Segmented list formatted as a list.....	269
1. Recognized units of measure.....	383

## List of Tables

1. An amaryllis.....	462
----------------------	-----

## List of Tables

2.1. Common DocBook effectivity attributes.....	30
4.1. Template %letter substitutions.....	67
5.1. The object map.....	78
5.2. The data map.....	78
5.3. The transformation map.....	85
1. Default keyboard navigation keys.....	135
1. Square numbers.....	277

## List of Examples

2.1. A contrived effectivity example.....	31
1. An example of char alignment.....	115
2. Tabular rendering.....	115
1. An example of media object accessibility.....	206
1. An example of char alignment.....	260
1. An example of table accessibility.....	276
1. An example of a cmdSynopsis.....	325
1. Several dates in ISO 8601 formats.....	426
1. The FAMILY-given name style.....	511
1. The first-last name style.....	512
1. The last-first name style.....	514

# **Part I. Introduction**



# Preface

These stylesheets are the third iteration of stylesheets for DocBook that I've written. I started working on the XSLT 1.0 Stylesheets for DocBook (<https://github.com/docbook/xslt10-stylesheets>) in the 90's, before XSLT 1.0 was even a W3C Recommendation. I started working on the XSLT 2.0 Stylesheets for DocBook (<https://github.com/docbook/xslt20-stylesheets>) just around the time when XSLT 2.0 became a W3C Recommendation. I wrote most of *DocBook xslTNG* just a month or so before the third anniversary of the XSLT 3.0 Recommendation.

Why did it take so long?

To answer that question, we need to reflect for a moment on XSLT and its place in the XML ecosystem. When XSLT arrived on the scene, we were near the peak of XML enthusiasm. Not only was XML supported everywhere, it was possible to imagine XSLT everywhere as well. Certainly, the presence of XSLT in the browser felt significant at the time.

The ubiquity of XML and the fact that XSLT was “just an XML vocabulary” may have contributed to another significant phenomenon: lots of users who did not self identify as programmers were learning to use XSLT and doing significant things with it.

There were other tools available for transforming markup at the time, and arguably some of them were better than XSLT, but they were programming languages and you had to be a programmer to use them. They were also mostly commercial applications not widely available to casual users.

XSLT was free, it was everywhere, and it was used by everyone, not “just” programmers. It was the clear winner then and remains the clear winner today in terms of markup transformation.

You could do a lot of things with *XSLT 1.0*. You could do a lot more things than you might at first even have thought possible. (In fact, you could do all things, but the Turing complete nature of XSLT isn't relevant here.) Some very common tasks, like grouping, were possible but difficult. Lots of very useful things were either not possible or required extensions: regular expressions, functions, date and time formatting, creating special characters in the output, to name just a few.

*XSLT 2.0* solved all of these problems (and more). Significantly, I think, all of these new features appealed directly to almost all users of XSLT 1.0. Everyone had encountered a grouping problem (building an index, for example). Everyone had wanted regular expression matching or date formatting. Lots of users wanted to write more sophisticated predicates (and many were willing to learn how to use functions to achieve that result).

## Preface

XSLT 3.0 arguably introduces larger and more dramatic features than XSLT 2.0 did. There are a bunch of new features designed to enable streaming processing; there are significant software engineering improvements: packaging, exception handling, and assertions; there are common programming language constructs like maps and arrays. There is also a selection of features inherited from updates to XPath (new functions, a subset of `let` syntax, and support for higher order functions, for example).

What's curious, I think, is that many of these features are probably less immediately appealing to many (most?) current users. XSLT 2.0 doesn't feel constraining in the same way that XSLT 1.0 did, and the features in XSLT 3.0 don't immediately and obviously solve problems that most users have.

Streaming, for example, is incredibly powerful and it's an important and significant milestone in markup processing. It makes it possible to solve whole classes of problems that were previously impossible to solve or required enormously expensive hardware. But my laptop will quite easily process a book full of complex markup that runs to hundreds of pages. I don't have any problems that require a streaming processor.

Likewise, packaging is useful and important. The *DocBook xslTNG* stylesheets should absolutely be a package. But that's not true of a lot of stylesheets. There might be software engineering benefit in making a package even for stylesheets that you don't intend to distribute, but that's more likely to appeal to people who think of what they're doing is programming.

Nevertheless, there are lots of good reasons to use XSLT 3.0 even if you are “only” transforming documents and even if you don't think of writing transformations as programming.

# Chapter 1. Overview

Before we get started, let's look at what the stylesheets do and where you can get them!

## 1.1. What do the stylesheets do?

The *DocBook xslTNG* stylesheets transform DocBook V5.x into HTML. The intent is that they support all of DocBook V5.2, including the DocBook Publishers elements. (The test suite report (<https://xsltng.docbook.org/report/>) gives a precise summary of the current state of coverage.) They will also process DocBook V4.x documents by first converting them (transforming source elements that have changed, adding the namespace, etc.) into 5.x documents.

Some parts of DocBook, especially the modeling parts, are very open-ended. One could, in principle, write a function synopsis for any programming language. The stylesheets are naturally going to support only a subset of languages out of the box. Every attempt has been made to make customizations easy where it's anticipated that they may be necessary.

The stylesheets can also be used to produce paged media such as PDF files. This works by having a slightly different HTML transformation initially, and then by further transforming the HTML so that it can be formatted with an appropriate CSS formatter to produce paged output. This is similar, but not the same as producing XSL Formatting Objects and then transforming those.

Producing other output formats, EPUB files, for example, is planned for the future but no specific schedule is promised. There are no plans at the moment to produce XSL FO stylesheets.

## 1.2. Getting the stylesheets

There are three ways to get the stylesheets: you can download the latest release from GitHub, you can get them from Maven, or you can clone the repository (<https://github.com/docbook/xslTNG>) and build them yourself. We'll cover the first two options in this chapter; *Chapter 6, Building the stylesheets* covers the last option in detail.

## 1.2.1. Download the latest release

The latest release is always available from the GitHub releases page (<https://github.com/docbook/xslTNG/releases>). At the time of publication, the latest release was version 2.1.0.

Distributed this way, you'll get a ZIP file that contains the stylesheets plus a number of ancillary files and tools. These are mostly covered in the next chapter *Chapter 2, Using the stylesheets*.

You can unzip the file anywhere that's convenient: in your home directory or in a system-wide location.

## 1.2.2. Getting the release from Maven

The latest release is always available from Maven (<https://search.maven.org/search?q=org.docbook.docbook-xslTNG>)<sup>1</sup>.

The group, artifact, and version ID for the latest release at the time of publication is:

org.docbook:docbook-xslTNG:2.1.0

If you're comfortable using Maven, I'm going to assume that's all you need to know.

The Maven distribution differs from the zip file in a couple of ways:

1. It doesn't bundle any of the dependencies. The ZIP file is more akin to an "uber" or "fat" jar (<https://stackoverflow.com/questions/11947037/what-is-an-uber-jar>); it includes (some of) the core dependencies so that it works out-of-the-box. Maven is designed to resolve dependencies, so that shouldn't be necessary here.
2. The Maven jar doesn't include the Python script or the extra resources (CSS and JavaScript files) because it doesn't seem like it would be convenient to extract them from the Maven jar (which will probably be installed deep in some repository hierarchy well out of sight). Consequently, you may want to download the distribution periodically as well.

---

<sup>1</sup> With the caveat that it sometimes takes a few hours for the releases to make their way from the registry onto the website. If you're chasing the very latest release and it's just been published, it may be available before it appears on the website.

# Chapter 2. Using the stylesheets

In principle, the stylesheets will run with any conformant XSLT 3.0 processor. For many users, that means Saxon (<http://saxonica.com/>). Although earlier versions may work, Saxon 10.1 or later is recommended.

In principle, the instructions for using the stylesheets are straightforward: using your XSLT 3.0 processor of choice, transform your DocBook source documents with the `docbook.xsl` stylesheet in the `xslt` directory of the distribution.

In practice, for most users, running the stylesheets will require getting a Java environment configured appropriately. For many, one of the most significant challenges is getting all of the dependencies sorted out. Modern software development, for better or worse, often consists of one library relying on another which relies on another, etc.

The *DocBook xsltNG* stylesheets attempt to simplify this process, especially for the “out of the box” experience by providing two convenience methods for running the stylesheets: a jar file with a `Main` class, and a Python script that attempts, among other things, to make sure all of the dependencies are available.

If you’re an experienced Java user, you may prefer to simply run the stylesheets directly with Java.

Irrespective of which method you choose, running the stylesheets is simply a matter of processing your input document “`myfile.xml`” with “`xslt/docbook.xsl`”. For example:

```
$ saxon myfile.xml -xsl:xslt/docbook.xsl -o:myfile.html
```

The exact path to `docbook.xsl` will vary, of course, but it’s in the `xslt` directory of the distribution.

### Note



The resulting HTML document contains references to CSS stylesheets and possibly JavaScript libraries. The output won't look as nice in your browser if those resources aren't available. They're in the `resources` directory of the distribution. A quick and easy way to see the results is simply to send the output to the `samples` directory from the distribution. The resources have already been copied into that directory. In the longer run, you'll want to make sure that they get copied into the output directory for each of your projects.

Alternatively, you can copy them to a web location of your choosing and point to them there. You can even point to them in the DocBook CDN (<https://cdn.docbook.org/release/xsltng/current/resources>), but beware that those are not immutable. The "current" version will change with every release and versioned releases will not persist indefinitely.

Change the `$resource-base-uri` to adjust the paths used in the output document.

Many aspects of the transformation can be controlled simply by setting parameters (see *I. Parameter reference*). It's also possible to change the transformation by writing your own customization layer (see *Chapter 3, Customizing the stylesheets*).

## 2.1. Using the Jar

The ZIP distribution includes a JAR file that you can run directly. That JAR file is `$ROOT/libs/docbook-xsltNG-version.jar` where "`$ROOT`" is whatever directory you chose to unzip the distribution into and `version` is the stylesheet version.

Assuming you unzipped the version 2.1.0 distribution into `/home/ndw/xsltng`, you can run the JAR like this:

```
java -jar /home/ndw/xsltng/libs/docbook-xsltNG-2.1.0.jar
```

## 2.1. Using the Jar

Let's try it out. Open a shell window and change to the samples directory, `/home/ndw/xsltng/samples` assuming you unzipped it as described above. Now run the java command:

```
$ java -jar ..libs/docbook-xslTNG-2.1.0.jar article.xml  
<!DOCTYPE html><html xmlns="http://www.w3.org/1999/xhtml">  
...more HTML here...  
<nav class="bottom"></nav></body></html>
```

That big splash of HTML was your first DocBook document formatted by the stylesheets! Slightly more usefully, you can save that HTML in a file:

```
$ java -jar ..libs/docbook-xslTNG-2.1.0.jar article.xml \  
-o:article.html
```

If you now open `article.html` in your favorite web browser, you'll see the transformed sample document which should look like *Figure 2.1, “The sample document: article.xml”*.

### Sample Article

This is a very simple DocBook document. It serves as a kind of “smoke test” to demonstrate that the stylesheets are working.

**Figure 2.1. The sample document: article.xml**

The JAR file, run this way, accepts the same command line options (<https://www.saxonica.com/html/documentation/using-xsl/commandline/>) as Saxon, with a few caveats:

No `-x`, `-y`, or `-r` options

The executable in the JAR file automatically configures Saxon to use a catalog-based resolver and points the resolver at a catalog that includes the files in the distribution.

No `-init` option

The *DocBook xslTNG* extension functions will be registered automatically.

Multiple `-catalog` options

You can repeat the `-catalog` option. All of the catalogs you specify will be searched before the default catalog.

### Default stylesheet

If you do not specify a stylesheet with the `-xsl` option, the `xslt/docbook.xsl` stylesheet will be used automatically.

## 2.2. Using the Python script

The ZIP distribution includes a Python script in the `bin` directory. This helper script is a convenience wrapper around Saxon. It sets up the Java classpath and automatically configures a catalog resolver and the DocBook extension functions.

### Important



The script requires the `click` and `pygments` packages, which you must install with pip before running the script. For example:

```
python3 -m pip install pygments=2.6.1 click
```

This script behaves much like the JAR file described in *Section 2.1, “Using the Jar”*. In particular, it accepts the same command line options (<https://www.saxonica.com/html/documentation/using-xsl/commandline/>) as Saxon, with the same caveats.

The significant feature of the Python script is that it will attempt to sort out the dependencies for you. It assumes that you’ve used Maven to install the package and its dependencies, so you’ll have to have installed Maven (<http://maven.apache.org/>). How you do that varies by platform, but your package manager probably has it.

The following command will assure that you’ve downloaded all of the necessary dependencies. You only have to do this once.

```
$ mvn org.apache.maven.plugins:maven-dependency-plugin:2.4:get \
-Dartifact=org.docbook:docbook-xslTNG:2.1.0
```

That might take a while.

The script will work through the dependencies that you have installed, and the things that they depend on, and construct a Java class path that includes them all.

## 2.2. Using the Python script

The script stores its configuration in `.docbook-xsltng.json` in your home directory.

Options passed to the script are processed as follows: the initial options, all introduced by two hyphens, are interpreted by this script; all the remaining options are passed directly to Saxon.

The script options are:

`--help`

Prints a usage message.

`--config:filename`

Use *filename* as the configuration file. The default configuration file is `.docbook-xsltng.json` in your home directory.

`--resources:dir`

This option will copy the resources directory (the CSS and JavaScript files) from the distribution into the directory where your output files are going, *dir*. If *dir* is not specified, the script attempts to work out the directory from the `-o` option provided to Saxon. If no directory is specified and it can't work out what the directory is, it does nothing.

`--java:javaexec`

Use *javaexec* as the Java executable. The default java executable is the first one on your `PATH`.

`--home:dir`

Use *dir* as the *DocBook xsltng* home directory. This should be the location where you unzipped the distribution. (You probably shouldn't change this.)

`--verbose`

Enables *verbose* mode; it prints more messages about what it finds.

`--debug`

Enables *debug* mode. Instead of running the transformation, it will print out the command that would have been run.

--

Immediately stop interpreting options. Everything that follows this option will be passed to Saxon, even if it begins with two hyphens.

## 2.3. Run with Java

Assuming you've organized your class path so that all of the dependencies are available (you may find that using a tool like Gradle or Maven simplifies this process), simply run the Saxon class.

For Saxon HE, the class is `net.sf.saxon.Transform`. For Saxon PE and EE, the class is `com.saxonica.Transform`.

## 2.4. Run with Docker

This is experimental.

The `docker` directory contains an experimental Dockerfile. Using docker allows you to isolate the environment necessary to run the *DocBook xslTNG Stylesheets* from your local environment.

Using Docker is a three step process. Step 0, you have to have installed Docker!

1. Build the docker image. In the `docker` directory, run the docker build command:

```
$ docker build -t docbook-xsltng .
```

The “`-t`” option provides a tag for the image; you can make this anything you want. There's a `VERSION` build argument if you want to build a particular release. For example,

```
$ docker build --build-arg VERSION=0.9.14 -t docbook-xsltng .
```

will build a Docker image for the 0.9.14 release of the stylesheets irrespective of the version in the Dockerfile.

2. Run the image, for example:

```
$ docker run docbook-xsltng samples/article.xml
```

## 2.5. Extension functions

If you chose a different tag when you built the image, use that tag in place of “`docbook-xsltng`” in the `run` command. Everything after the container tag becomes options to the `docbook` Python script.

### Note



The context the script runs in is *inside* the container. It can't for example, see your local filesystem. The example above works because the distribution is unpacked inside the container. So the `article.xml` document isn't the one on your local filesystem.

You can use the Docker facilities for mounting directories to change what documents the script can access. For example:

```
$ docker run -v /tmp:/output -v /path/to/samples:/input \
    docbook-xsltng /input/article.xml chunk=index.html \
    chunk-output-base-uri=/output/
```

Assuming that the “samples” directory in the distribution is located at `/path/to/samples`, this will chunk the `article.xml` sample document that the script sees in `/input` (which is where you mounted `samples`) and it will write the output to `/output` (which is where you mounted `/tmp`).

When the script finishes, the chunked output will be in `/tmp`.

### Tip



If you choose to use Docker, you don't have to rebuild the container everytime a new stylesheet release occurs. You can simply mount the new `xslt` directory into the container like any other directory.

## 2.5. Extension functions

The stylesheets are distributed with several extension functions:

`ext:cwd()`

Returns the “current working directory” where the processor is running.

`ext:image-properties()`

Returns basic properties of an image, width and height.

`ext:image-metadata()`

Returns much more comprehensive image properties and understands far more image types than `ext:image-properties()`. Requires the *metadata-extractor* libraries.

`ext:pygmentize()`

Runs the external *Pygments* processor on a verbatim listing to add syntax highlighting.

`ext:pygmentize-available()`

Returns true if the external *Pygments* processor is available on the current system.

`ext:xinclude()`

Performs *XInclude* processing. This extension supports the basic XPointer schemes, *RFC 5147* fragment identifiers, and *search*, a scheme that supports searching in text documents.

`ext:validate-with-relax-ng()`

Performs RELAX NG validation.

At the time of this writing, all of these extension functions require Saxon 10.1 or later. Make sure that the `docbook-xslng-version.jar` file is on your class path and use the Saxon `-init` option to load them:

`-init:org.docbook.xslng.extensions.Register`

### 2.5.1. Extension function debugging

When an extension function fails, or produces result other than what you expect, it can be difficult sometimes to work out what happened. You can enable debugging messages by setting the the system property `org.docbook.xslng.verbose`.

Setting the property to the value “`true`” enables all of the debugging messages. For a more selective approach, set it to a comma separated list of keyword values.

## 2.6. “Chunked” output

The following keywords are recognized:

`registration`

Enables messages related to function registration.

`image-properties`

Enables messages related to image properties.

`image-errors`

Enables messages related to image properties, but only when the function was unable to find the properties or encountered some sort of error condition.

`pygmentize-show-command`

Enables a message that will show the pygmentize command as it was run.

`pygmentize-show-results`

Enables a message that will show the output of the pygmentize command, before it is processed by the function.

`pygmentize-errors`

Enables messages related to errors encountered attempting to highlight listings with pygmentize.

## 2.6. “Chunked” output

Transforming “`myfile.xml`” with `docbook.xsl` usually produces a single HTML document. For large documents, books like this one for example, it’s sometimes desirable to break the input document into a collection of web pages. You can achieve this with the *DocBook xsltNG Stylesheets* by setting two parameters:

`$chunk`

This parameter should be set to the name that you want to use for the first, or top, page of the result. The name “`index.html`” is a common choice.

### \$chunk-output-base-uri

This parameter should be set to the absolute path where you want to use as the base URI for the result documents, for example `/home/ndw/output/guide/`.

#### Note



The trailing slash is important, this is a URI. If you specify only `/home/ndw/output/guide`, the base URI will be taken to be `/home/ndw/output/`, and the documents won't have the URIs you expect.

This output URI has nothing to do with where your documents are ultimately published and the documents themselves won't contain any references to it. It simply establishes the root of output. If you're running your XSLT processor from the command line, it's likely that the documents will be written to that location. If you're running an XProc pipeline, it simply controls the URIs that appear on the secondary result port.

Many aspects of chunking can be easily customized. A few of the most relevant parameters and templates are:

### \$chunk-include and \$chunk-exclude

Taken together, these two parameters determine what elements in your source document will be considered “chunks” in the output.

### \$persistent-toc

If this parameter is true, then a JavaScript “fly-out” table of contents will be available on every page.

### \$chunk-nav

This parameter, discussed more thoroughly in *Section 2.6.1, “Keyboard navigation and speaker notes”* enables keyboard navigation between chunks.

### t:top-nav and t:bottom-nav

These templates control how the top-of-page and bottom-of-page navigation aids are constructed.

## 2.6.1. Keyboard navigation and speaker notes

### 2.6.1. Keyboard navigation and speaker notes

If the `$chunk-nav` parameter is true, a reference to an additional JavaScript library will be included in the resulting pages. This library supports keyboard navigation between the pages. The navigation keys are described in the parameter reference page.

There is an additional customization layer (`xslt/speaker-notes.xsl`) provided for adding speaker notes to the pages. This is provided both as an example of a customization layer and because the author finds it convenient.

If you use the speaker notes customization layer, the any top-level element in a chunk with the role “`speaker-notes`” will be suppressed from the default presentation. If you press “`S`” on the page, then you’ll get a “speaker notes” view of the page.

This can be combined with another extension, the use of browser local storage, to create a simple presentation system. Add this `meta` tag to the `info` element of your document:

```
<meta xmlns="http://www.w3.org/1999/xhtml"  
      name="localStorage.key" content="keyname"/>
```

That will cause the pages to keep track of their location using the “`keyname`” property in local storage. This is important because it enables the following trick:

1. Configure keyboard navigation, speaker notes, and local storage in your document.
2. Arrange for your document to be served up from a web server. You can do this by running one locally or by putting the documents on a web server somewhere else.
3. Open up the main page of your document in a browser.
4. Open up a second browser *window* pointing to the same page. Navigate back and forth between the pages. You should see that the two windows stay in sync.
5. Now press “`S`” in one of the windows and navigate around. You should see that the two windows stay in sync and that your speaker notes are consistently presented in one of the windows.

I often use this trick when I’m giving presentations. I can project the slides in one browser window and keep the other browser window on my laptop. This allows me to see my notes while easily projecting the “real” content.

## 2.7. Effectivity attributes and profiling

When documenting computer hardware and software systems, it's very common to have different documentation sets that overlap significantly. Documentation for two different models of network router, for example, might differ only in a few specific details. Or a user guide aimed at experts might have a lot in common with the new user guide.

### 2.7.1. Effectivity

There are many ways to address this problem, but one of the simplest is to identify the “effectivity” of different parts of a document. Effectivity in this context means identifying the parts of a document that are effective for different audiences.

When a document is formatted, the stylesheets can selectively include or omit elements based on their configured effectivity. This “profiled” version of the document is the one that’s explicitly targeted to the audience specified.

DocBook supports a wide variety of common attributes for this purpose:

**Table 2.1. Common DocBook effectivity attributes**

Attribute	Nominal effectivity axis
arch	The architecture, Intel or AMD
audience	The audience, operations or development
condition	Any condition (semantically neutral)
conformance	The conformance level
os	The operating system, Windows or Linux
outputformat	The output format, print or online
revision	The revision, 3.4 or 4.0.
security	The security, secret or top-secret
userlevel	The user level, novice or expert
vendor	The vendor, Acme or Yoyodyne
wordsize	The word size, 32 or 64 bit

In addition, the stylesheets support profiling on several common attributes that are not explicitly for effectivity: `xml:lang`, `revisionflag`, and `role`.

**Note**

DocBook places no constraints on the values used for effectivity and the stylesheets don't either. You're free to use "cat" and "dog" as effectivity values in the `wordsize` attribute, if you wish. The further you deviate from the nominal meaning, the more important it is to document your system!

Consider *Example 2.1, “A contrived effectivity example”*.

```
<para>This is an utterly contrived example of  
some common text. Options are introduced with the  
<phrase os="windows">/</phrase>  
<phrase os="mac;linux">-</phrase> character.</para>
```

**Example 2.1. A contrived effectivity example**

If this document is formatted with the `$profile-os` parameter set to "windows", it will produce:

This is an utterly contrived example of some common text. Options are introduced with the / character.

If "mac" or "linux" is specified, it will produce:

This is an utterly contrived example of some common text. Options are introduced with the - character.

**Important**

If the document is formatted without any profiling, *all* of the versions will be included:

This is an utterly contrived example of some common text. Options are introduced with the / - character.

That is unlikely to work well.

## 2.7.2. Profiling

The profiling parameters are applied to every document: `$profile-arch`, `$profile-audience`, `$profile-condition`, `$profile-conformance`, `$profile-lang`, `$profile-os`, `$profile-outputformat`, `$profile-revision`, `$profile-revisionflag`, `$profile-role`, `$profile-security`, `$profile-userlevel`, `$profile-vendor`, and `$profile-wordsize`. Each of these values is treated as a string and broken into tokens at the `$profile-separator`.

For every element in the source document:

- If it specifies a value for an effectivity attribute, the value is split into tokens at the `$profile-separator`.
- If the corresponding profile parameter is not empty, then the element is discarded unless at least one of the tokens in the profile parameter list is also in the effectivity list.

In practice, elements that don't specify effectivity are always included and profile parameters that are empty don't exclude any elements.

## 2.7.3. Dynamic profiling

Dynamic profiling is a feature that allows you to profile the output of the stylesheets according to the runtime values of stylesheet parameters. You can, for example, produce different output depending on whether or not chunking is enabled or JavaScript is being used for annotations.

To enable dynamic profiling, set the `$dynamic-profiles` parameter to “true”.

In the interest of performance, security, and legibility, dynamic profiles don't support arbitrary expressions. You can use a variable name by itself, `$flag`, which tests if that variable is true, or you can use a simple comparison, `$var=value` which tests if (the string value of) `$var` has the value `value`. (If `$var` is a list, it's an existential test.) You also can't use boolean operators or any other fancy expressions.

If you really need to have a dynamic profile based on some arbitrary condition, you can do it by making a customization layer that stores that computation in a variable and then testing that variable in your dynamic profile.

### Backwards incompatibility

This is slightly backwards incompatible in that profile values that begin with a dollar sign are now interpreted differently. This is only true if dynamic profiling is enabled.

An element with dynamic profiling will be published if none of its profile expressions evaluate to false. This is slightly different from the ordinary profiling semantic which publishes the element if any of its values match.

## 2.8. Syntax highlighting

Program listings and other verbatim environments can be “syntax highlighted”, that is, the significant tokens in the listing can be colored differently (keywords in red, quoted strings in blue, that sort of thing).

The default syntax highlighter is *Pygments*, an external Python program. This has the advantage that the highlighted listing is available to the stylesheets. The stylesheets can then render line numbers, call outs, and other features.

But running an external program for every verbatim environment requires *having* the external program and also, if there are many verbatim environments, may slow down the formatting process

An alternative is to use a JavaScript highlighter in the browser such as *highlight.js* or *Prism*. This approach has no impact on formatting and doesn’t require an external process. However, it means the *xsltTNG Stylesheets* have no control over the process. Most of the verbatim options only apply when Pygments is used.

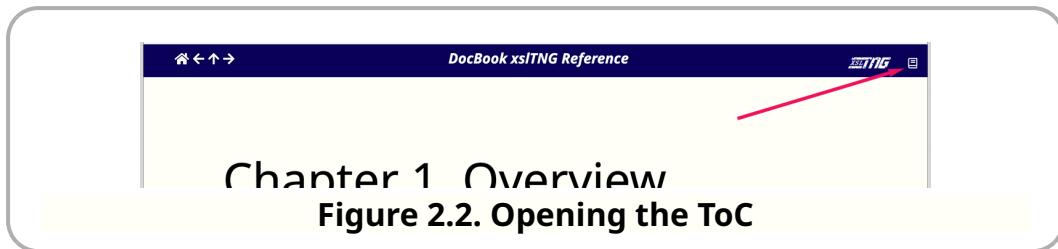
The choice of syntax highlighter is determined by the `$verbatim-syntax-highlighter` parameter.

## 2.9. Persistent table of contents

The persistent Table of Contents (ToC) provides a full ToC for an entire document accessible from each chunked page.

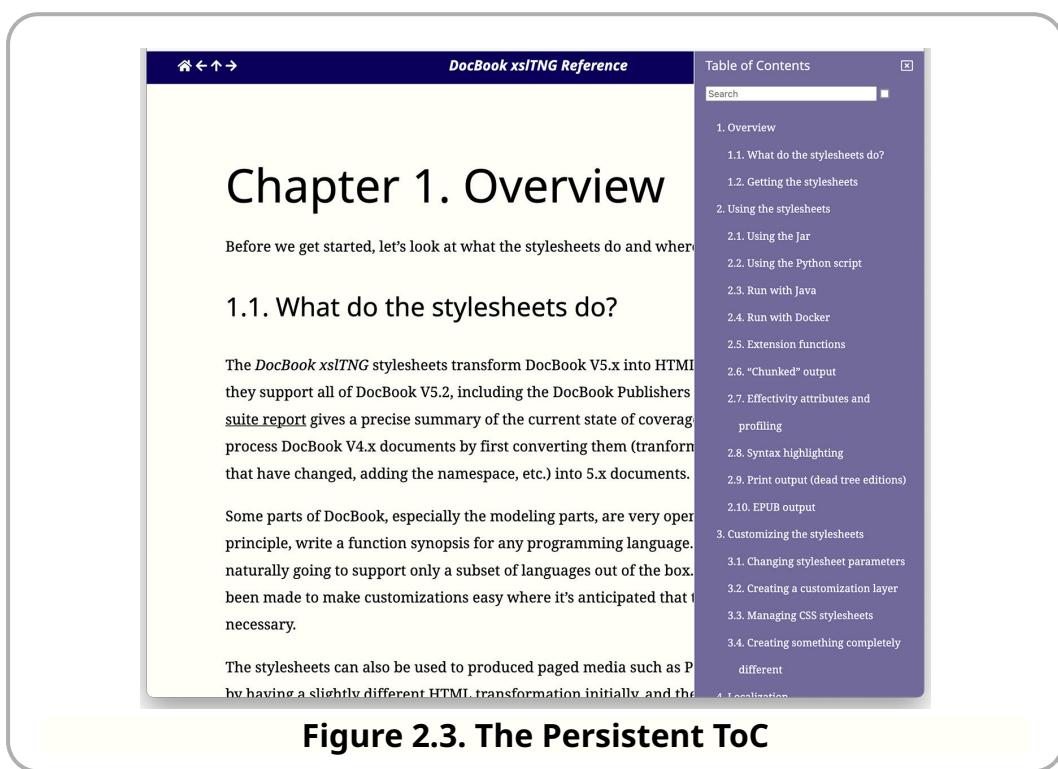
The ToC is accessed by clicking on the “book” icon in the upper right corner of the page as shown in *Figure 2.2, “Opening the ToC”*.

## Chapter 2. Using the stylesheets



The icon and other aspects of the style can be changed by providing `$persistent-toc-css`.

Once open, the ToC is displayed. A long ToC will be scrolled to the location of the current page in the document as shown in *Figure 2.3, “The Persistent ToC”*.



**Figure 2.3. The Persistent ToC**

The persistent ToC popup is transient by default, meaning that it will disappear if you use it to navigate to a different page. If you open the popup by “shift-clicking” on it, the ToC will persist until you dismiss it. This can also be accomplished by selecting the check box in the ToC. The presence of the search bar is controlled by the `$persistent-toc-search` parameter.

### 2.9.1. Persistent ToC data

The data used by the persistent ToC can be stored in a separate file or stored in each chunk. This is controlled by the `$persistent-toc-filename`.

### 2.9.1. Persistent ToC data

1. If chunking is enabled and the `$persistent-toc-filename` parameter is non-empty, it's used as a filename and a single copy of the ToC will be saved in that file.

The benefit of this approach is that the HTML chunks are smaller. If the persistent ToC is written into every chunk, the size of each HTML chunk increases in proportion to the size of the ToC. For a large document with lots of small pages, this can be a significant percentage of the overall size.

There are two disadvantages:

- a. This will not work if the documents are accessed with `file:` URIs: you must use `http` (and in some environments, perhaps `https`) to load the documents. The browser will (quite reasonably) not allow JavaScript to load documents from the filesystem.
  - b. Also, with this approach, opening the ToC requires another document to be loaded into the browser. For a large ToC, this can introduce visible latency, although browser caching tends to reduce that after the document has been loaded once.
2. If the `$persistent-toc-filename` parameter is the empty sequence, a copy of the ToC is stored in each chunk.

#### Note



When stored in each chunk, the Table of Contents is secreted away in a `script` element so that it will be ignored by screen readers and other user agents that don't support JavaScript or CSS.

The benefit of this approach is that it requires no additional document to be loaded and will work even if the documents are loaded with `file` URIs.

The disadvantage of this approach is that it increases the size of each chunk. Whether that matters depends on the size of the ToC, the relative size of the chunks, bandwidth and other constraints.

3. If chunking is *not* being used, there will only be one HTML result and the ToC will always be stored in that chunk.

## 2.10. On-page table of contents

Documents come in many shapes and sizes. Consequently, there are a variety of navigation mechanisms available. For long documents, such as books, a Table of Contents (ToC) is traditional (as are indexes). For web presentation, long documents may be broken into chunks, for example at the chapter level. In this case, header and footer navigation between chunks is almost always available. For large documents a “persistent ToC” can enable quick navigation from any chunk.

You can also enable an on-page ToC. The on-page ToC provides a navigation mechanism for sections within a page. By default, it appears on the right of the page if the window is wide enough to comfortably display it next to the main body.

The current implementation requires JavaScript. In fact it is not constructed from the DocBook markup, but instead from the HTML markup when the page is rendered. To be precise, the ToC is constructed from HTML `section` elements that begin with a `header` that contains an `h1` ... `h6` element. It is therefore either a bug or a feature, depending on your perspective, that a customization layer that changes how sections are marked up will change what appears in the ToC. If you simply wish to suppress a particular section from appearing in the ToC, add `nopagetoc` to its `class` attribute.

Several parameters control presentation and formatting of the on-page ToC.

### \$pagetoc-elements

A list of the names of the elements (technically, the classes of the sections) that should get an on-page ToC. It’s empty by default (meaning no such ToC is rendered). For the standard presentation of this guide, the list is set to `preface chapter appendix refentry`. (The sneaky among you may wonder if you could simply set it to “`component`” because that class is used for all those elements; “Yes. Yes, you could.”)

### \$pagetoc-dynamic

Determines whether or not the ToC is “dynamic”. Inspired by Kevin Drum’s table of contents progress animation (<https://kld.dev/toc-animation/>), the ToC keeps track of the reader’s location in the main view and highlights the corresponding sections in the ToC (albeit without the clever SVG animation of the original).

Set this parameter to false if you find the animation distracting. (If the animation is enabled, a control is provided to let the reader turn it off, in case *they* find it distracting.)

## 2.10. On-page table of contents

### \$pagetoc-js

This is the JavaScript that implements the on-page ToC. Changing this parameter allows you to replace it with JavaScript of your own invention.

### CSS

There is no `pagetoc-css` parameter; the CSS is integrated into the standard CSS. You can find it in the `pagetoc.scss` file in the repository if you want to change the presentation. (Don't change that file, simply add overriding rules later in the cascade.)

There is also a JavaScript API that you can use to control some features of the presentation. This is done by adding a `DocBook` property to the browser's `window` object. The value of the `DocBook` property should be a map. To control the on-page ToC, add a `pagetoc` property to the `DocBook` map. The value of this property must also be a map.

The properties of the `pagetoc` map can be used to change the display:

#### decorated

This is the markup used for the user-control on the on-page ToC when the ToC is decorated. The default value is “

#### plain

This is the markup used for the user-control on the on-page ToC when the ToC is plain (not decorated). The default value is “

#### hidden

This is the markup used for the user-control on the on-page ToC when the ToC is hidden. The default value is “

#### nothing\_to\_reveal

This property controls how the on-page ToC is rendered if there are no additional sections to be revealed. It can have the value “`hide`”, to hide the ToC, “`plain`” to make its presentation plain in this case, or “`decorated`” to use the decorated style. The ToC will not appear if there are no sections.

To use the JavaScript API, make sure your assignments to the `DocBook` object are performed *before* the on-page ToC JavaScript is executed.

## 2.11. Print output (dead tree editions)

Formatters, the tools that turn markup of any sort into aesthetically pleasing (or even passably acceptable) printed pages are fiendishly difficult to write.

In the XML space, there have been a number of standards and vendor-specific solutions to this problem. The current standards are *XSL FO* and *CSS*.

At present, the *DocBook xslTNG Stylesheets* are focused on CSS for print output. There's a customization layer that produces "paged-media-ready" HTML that can be processed with a CSS formatter such as *AntennaHouse* or *Prince*.

To get print output, format your documents with the `print.xsl` stylesheet instead of the `docbook.xsl` stylesheet. The additional cleanup provided by `print.xsl` assures that footnotes, annotations, and other elements will appear in the right place, and with reasonable presentation, in the printed version.

The resulting HTML document can be formatted directly with a CSS paged-media formatter.

## 2.12. EPUB output

The *DocBook xslTNG Stylesheets* will produce output designed for EPUB(3) if you use the `epub.xsl` stylesheet instead of `docbook.xsl`. This is new in version 1.11.0 and may be incomplete. The output conforms to EPUBCheck (<https://www.w3.org/publishing/epubcheck/>) version 3.2.

Producing an EPUB file is a slightly complicated process. You must produce (X)HTML that conforms to strict requirements, you must produce a media type document containing a specific text string, you must produce a manifest that identifies all of the content including all the images, stylesheets, fonts, etc, and you must finally create a ZIP archive (with some special consideration as well).

The stylesheets can only do part of this process. In some future release where we use, for example, an XProc 3.0 pipeline, it may be practical to do more.

When you run the EPUB stylesheet, the principle result document is the media type document. This has two useful side effects: first, it establishes the output base URI from which all the relative documents can be created, and second, if you fail to process some element in the input, you're likely to get extra text characters in the principle result document. That will cause tools to reject the EPUB and draw your attention to the error.

### 2.12.1. Adding metadata

The stylesheets also produce the META-INF files and the OPS directory containing the document parts and the manifest.

There are two parameters specific to EPUB:

`pub-id`

This is the unique identifier for your EPUB. If you don't specify one, a random one will be generated for you.

`manifest-extra`

This is a URI. If it's specified, then it must be an XML document and that will be added to the EPUB manifest. This is how you can add links to media and other resources that the stylesheets don't know about.

## 2.12.1. Adding metadata

You can add elements to the `info` element of the root element of your document to add metadata to your EPUB files. Elements in the Dublin Core namespace will be copied through. You can also add the elements `meta` and `link` in the special namespace `http://docbook.org/ns/docbook/epub`.

## 2.12.2. EPUB in action

The Getting Started (<https://github.com/docbook/getting-started/>) project has been updated to show how to create EPUB from a book. The project has support for dealing with external media, fonts, and constructing the final ZIP file.



# Chapter 3. Customizing the stylesheets

In many circumstances, the stylesheets can be used “out of the box” without any customization. But sometimes you may need to change the formatting of certain elements. One common reason is to change the formatting of title pages or navigational features. In other cases, it may be to support local extensions to DocBook or simply to change the markup to support a particular use case.

Three approaches are possible, with increasing degrees of effort: changing stylesheet parameters, creating your own customization layer, or making broader changes to the stylesheet’s implementation.

The subject of broader implementation changes is the subject of *Chapter 5, Implementation details*. In this chapter, we’ll look at the easier options.

## 3.1. Changing stylesheet parameters

The *DocBook xslTNG Stylesheets* define a lot of parameters. They are all described in *I. Parameter reference*. If the change you want to make has already been parameterized, you may be able to achieve your goal simply by setting a parameter at runtime.

For example, if you want to change the formatting of dates and times in `date` elements, you can simply change the date and time formatting parameters. Similarly, if you want to change the numeration style of ordered lists, you can simply change the ordered list item numeration parameter.

These changes can be accomplished by simply passing the new values to the processor, on the command line or in a configuration file, for example. You do not have to write any XSLT to make these changes.

Parameter values apply to the entire document processed by the stylesheets. In some cases, you may wish to change the presentation of just one or small number of elements. This can often be accomplished with a db processing instruction in the source document itself. These customizations can also be accomplished without writing any XSLT.

If you want to make a change that isn’t supported by a parameter, or an ad hoc exception that doesn’t have a supporting processing instruction, you will have to write a customization layer. (You are invited to submit an issue with your use case if you think it would be of general interest.)

You may also find it convenient to write a customization layer if you want to change several parameters and you find it inconvenient to pass them all to the processor on every invocation.

## 3.2. Creating a customization layer

A customization layer is simply an XSLT stylesheet that you write which extends the DocBook stylesheets. The simplest<sup>1</sup> customization layer is:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <xsl:stylesheet
3      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4      xmlns:db="http://docbook.org/ns/docbook"
5      xmlns:xs="http://www.w3.org/2001/XMLSchema"
6      xmlns="http://www.w3.org/1999/xhtml"
7      exclude-result-prefixes="db xs"
8      version="3.0">
9
10 <!-- This href has to point to your local copy
   of the stylesheets. -->
11 <xsl:import href="docbook/xslt/docbook.xsl"/>
12
13 </xsl:stylesheet>
```

This customization doesn't do anything. But you can, for example, redefine parameters if you wish:

---

<sup>1</sup> Ok, technically, this stylesheet has a couple of namespace references that aren't strictly necessary so it could be a teeny bit simpler, but you'll need those declarations (and more!) if you want to do anything useful.

### 3.2. Creating a customization layer

```
1 <?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:db="http://docbook.org/ns/docbook"
5    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.w3.org/1999/xhtml"
    exclude-result-prefixes="db xs"
    version="3.0">

10 <xsl:import href="docbook/xslt/docbook.xsl"/>

<xsl:param name="orderedlist-item-numeration"
    select=""1""/>

15 <xsl:param name="date-dateTime-format"
    select=""[D01] [MNn,*-3] [Y0001]
        at [H01]:[m01]""/>

</xsl:stylesheet>
```

This will have the effect of running the DocBook stylesheets with those two parameters set as specified.

If you want to change the HTML output for an element, you can write a template for that element in your customization layer. Consider this DocBook document:

```
1 <?xml version="1.0" encoding="utf-8"?>
2   <article xmlns="http://docbook.org/ns/docbook"
3     version="5.1">
4     <info>
5       <title>Sample Document</title>
6       <date>2020-07-05</date>
7     </info>
8
9     <para>This is a sample <productname>DocBook</productname>
10    document.</para>
11
12   </article>
```

Suppose that you decided you wanted to have the `productname` element link automatically to the vendor webpage.

### Important



The *DocBook xslTNG Stylesheets* process *all* DocBook elements in the `m:docbook` mode. This is different from previous XSLT stylesheets for DocBook which simply used the default mode.

You must either specify a default mode in your customization layer or remember to specify the mode on match templates and template applications. If you forget the mode, you'll get unexpected results!

One way to do that would be to redefine the template that processes the `productname` element:

### 3.2. Creating a customization layer

```
1 <?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:db="http://docbook.org/ns/docbook"
5     xmlns:m="http://docbook.org/ns/docbook/modes" ①
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.w3.org/1999/xhtml"
    exclude-result-prefixes="db m xs" ②
    version="3.0">
10
<xsl:import href="docbook/xslt/docbook.xsl"/>

<xsl:param name="orderedlist-item-numeration"
    select=""1""/>
15
<xsl:param name="date-dateTime-format"
    select="[D01] [MNn,*-3] [Y0001]
        at [H01]:[m01]" />

20 <xsl:template match="db:productname"
    mode="m:docbook"> ③
    <xsl:variable name="name"
        select="normalize-space(.)"/>
④
25   <xsl:variable name="url" as="xs:string?">
        <xsl:choose>
            <xsl:when test="$name='DocBook'">
                <xsl:sequence select="https://docbook.org/" />
            </xsl:when>
30            <xsl:when test="$name='DocBook xslTNG Stylesheets'">
                <xsl:sequence select="https://xsltng.docbook.org/" />
            </xsl:when>
            <xsl:when test="$name='Wikipedia'">
                <xsl:sequence select="https://wikipedia.org/" />
35            </xsl:when>
            <xsl:otherwise>
                <!-- Unrecognized -->
```

- ① All of the DocBook elements are processed in the “`m:docbook`” mode.
- ② Remember to exclude all the namespaces you declare so that they don’t wind up scattered about in your HTML.
- ③ I repeat, all of the DocBook elements are processed in the “`m:docbook`” mode. I expect failure to declare this mode is going to be a common error.
- ④ Yes, this whole listing is rather cramped. I’m trying to make it all narrow enough to fit in the display without overflowing the margins.
- ⑤ Calling `xsl:next-match` invokes the underlying processing. The effect of this template is to wrap an HTML “`a`” around the default processing for `productname`.

It’s worth pointing out that if the tag has an `xlink:href` attribute, that will generate an HTML `a` as well. A more robust stylesheet would check for that, but I’m trying to keep the example simple.

### 3.3. Managing CSS stylesheets

The HTML that the *DocBook xslTNG* stylesheet produce is intended to be clean, robust markup for styling with CSS. Exactly how you control which stylesheet links are produced has changed several times. The current scheme is this:

1. If syntax highlighting is enabled, a link to the `$verbatim-syntax-highlight-css` stylesheet is included.
2. If `$persistent-toc` is true a link to the `$persistent-toc-css` stylesheet is included.
3. If `$use-docbook-css` is true, links to the standard DocBook stylesheets are included. Those stylesheets are `docbook.css` (for all media), `docbook-screen.css` (for screen media), and `docbook-page-setup.css` and `docbook-paged.css` (for print media).
4. The DocBook element that is the context element when the HTML `head` is being generated is processed in the `m:html-head-links` mode. By default, that template does nothing, but you can change that in a customization layer.
5. If any CSS stylesheets are defined in `$user-css-links`, they are included.

6. The DocBook element that is the context element when the HTML `head` is being generated is processed in the `m:html-head-last` mode. By default, that template does nothing, but you can change that in a customization layer.

## 3.4. Managing media

References to external media through `imagedata`, `videodata`, `audiodata`, and even `textdata` can be tricky to manage. On the one hand, it's most convenient if the URIs in the source documents point to the actual media files. This allows extensions, like the image properties extension function, to access the files. At the same time, the references generated in the HTML have to point to the locations where they will be published. It is often, but not always, the case that the authoring structures and the publishing structures are the same.

The stylesheets are regularly tested against five possible arrangements: three where the media are stored in locations relative to the XML files and two where the media are stored in a separate hierarchy. These are unimaginative named “mo-1”, “mo-2”, “mo-3”, “mo-4”, and “mo-5”. You can find them in the `src/test/resources/xml` (<https://github.com/docbook/xsltNG/tree/main/src/test/resources/xml>) hierarchy in the repository.

mo-1

All of the XML files are in a single directory, the media are in the same hierarchy. Media references in the source use relative URIs to refer to the underlying media: `preface.xml` refers to the “this is a test” audio clip as `media/this-is-a-test.mp3`.

mo-2

The XML files are in different directories (this changes the base URI of the media elements). The media are in the same hierarchy. Media references in the source use relative URIs to refer to the underlying media: `front/preface.xml` refers to the “this is a test” audio clip as `..../media/spinning-top.mp4`.

mo-3

The XML files are in different directories, but the structure is deeper. This scenario represents the case where there might be multiple books, each with their own media, but also a shared media folder “above” the book hierarchies. The media are in the same hierarchy, but some are “above” the book. Media references in the source use relative URIs to refer to the underlying media: `book/front/preface.xml` refers to the “this is a test” audio clip as `../../../../media/spinning-top.mp4`.

mo-4

The XML files are still in different directories, but the significant change here is that the media are in their own hierarchy. Media references in the source use URIs relative to the root of *that* hierarchy: `book/front/preface.xml` refers to the “this is a test” audio clip as `spinning-top.mp4`.

mo-5

The XML files are in different directories and the media are in their own hierarchy. What’s different here is that the media hierarchy is *further* subdivided by media type. Media references in the source use URIs relative to the root of media hierarchy *without* the media type: `book/front/preface.xml` still refers to the “this is a test” audio clip as `spinning-top.mp4`, but this time it is found in `media/mp4/spinning-top.mp4` rather than directly in `media`.

For each arrangement, we look at five possible output structures:

1. A single HTML document with the media in the same relative locations as the sources.
2. A single HTML document with the media in a single `media` subdirectory.
3. “Chunked” HTML output with the media in the same relative locations as the sources.
4. “Chunked” HTML output with the media in custom locations. (This is especially tricky for the “mo-5 case because there are two kinds of customization involved.)
5. “Chunked” HTML output with the media in a single `media` subdirectory.

The list below gives a brief summary of the parameters used to achieve the desired results for each combination of input and output arrangements.

**Note**



Remember that in each case, the questions are: can the stylesheets find the media files to query them and are the correct HTML references produced? Actually copying the media files from where they are in the source system to where they need to be in the HTML is “not our problem.”

### 3.4. Managing media

mo-1, mo-2, and mo-3 / scenario 1

No parameters are needed, this combination works correctly with the defaults.

mo-1, mo-2, and mo-3 / scenario 2

```
mediaobject-output-base-uri = "media/"  
mediaobject-output-paths = "false"
```

The output base URI is relative to the “root” of the HTML result. Setting the output paths to “false” removes intermediate hierarchy from the image references.

mo-1, mo-2, and mo-3 / scenario 3

```
chunk = "index.html"  
chunk-output-base-uri = "/path/to/output/location/"
```

These parameters aren’t related to media objects, they just tell the stylesheets how and where to “chunk” the output.

mo-1, mo-2, and mo-3 / scenario 4

```
chunk = "index.html"  
chunk-output-base-uri = "/path/to/output/location/"
```

This combination is really the same as the previous except that it uses a custom stylesheet ([https://github.com/docbook/xslTNG/blob/main/src/test/resources/mo\\_1\\_test\\_4.xsl](https://github.com/docbook/xslTNG/blob/main/src/test/resources/mo_1_test_4.xsl)) with a template in the `m:mediaobject-output-adjust` mode to add an extra level of hierarchy to the output URIs. This is just an example of arbitrary, custom processing.

mo-1, mo-2, and mo-3 / scenario 5

```
chunk = "index.html"  
chunk-output-base-uri = "/path/to/output/location/"  
mediaobject-output-base-uri = "media/"  
mediaobject-output-paths = "false"
```

The output base URI is relative to the “root” of the HTML result. Setting the output paths to “false” removes intermediate hierarchy from the image references.

### mo-4 / scenario 1

```
mediaobject-input-base-uri = "../media/"
```

The input base URI will be made absolute relative to the base URI of the input document, so it's often convenient to specify it as a relative URI. It's equally possible to specify it as an absolute URI.

### mo-4 / scenario 2

```
mediaobject-input-base-uri = "../media/"  
mediaobject-output-base-uri = "media/"  
mediaobject-output-paths = "true"
```

This example has two images with the same name in different directories, so it's necessary to preserve the output paths.

### mo-4 / scenario 3

```
chunk = "index.html"  
chunk-output-base-uri = "/path/to/output/location/"  
mediaobject-input-base-uri = "../media/"
```

This is the combination of chunking and a single media directory.

### mo-4 / scenario 4

```
chunk = "index.html"  
chunk-output-base-uri = "/path/to/output/location/"  
mediaobject-input-base-uri = "../media/"
```

This combination is really the same as the previous except that it uses a custom stylesheet ([https://github.com/docbook/xslTNG/blob/main/src/test/resources/mo\\_1\\_test\\_4.xsl](https://github.com/docbook/xslTNG/blob/main/src/test/resources/mo_1_test_4.xsl)) with a template in the `m:mediaobject-output-adjust` mode to add an extra level of hierarchy to the output URIs. This is just an example of arbitrary, custom processing.

### 3.5. Controlling numeration

mo-4 / scenario 5

```
chunk = "index.html"  
chunk-output-base-uri = "/path/to/output/location/"  
mediaobject-input-base-uri = "../media/"  
mediaobject-output-base-uri = "media/"  
mediaobject-output-paths = "true"
```

This is effectively scenario 2 with chunking.

mo-5 / scenarios 1-5

The “mo-5” scenarios are all the same as the “mo-4” scenarios with the addition of one more parameter:

```
mediaobject-grouped-by-type = "true"
```

In each case, this adds the extra “media object type” level to the URI path.

If you download the source repository, you can see these combinations in action with the build targets “`mo_number_test_scenario`”, for example, run:

```
./gradlew mo_3_test_2
```

to see the results of processing “mo-3” in scenario 2. The output will be in the `build/actual` directory. The build target `all_mo_tests` will run them all.

## 3.5. Controlling numeration

Numeration refers to the process(es) by which sets, books, divisions, components, sections, and formal objects are numbered. There are three separate aspects to numeration: what’s numbered, where does numbering begin, and does the number inherit from its ancestors.

Consider this book:

```

1 <book>
  <title>Book title</title>
  <part>
    <title>Part title</title>
5   <chapter>
    <title>Chapter title</title>
    <para>...</para>
    </chapter>
  </part>
10 <part>
    <title>Another part title</title>
    <chapter>
      <title>Another chapter title</title>
      <para>...</para>
15   </chapter>
    <chapter>
      <title>Yet another chapter title</title>
      <para>...</para>
    </chapter>
20   </part>
</book>

```

Let's suppose that parts are numbered "I" and "II". (The number format is controlled by the localization, see *Chapter 4, Localization*.) If chapter numbering begins at the book level, those chapters will be numbered "1", "2", and "3". If chapter numbering begins at the division level (the `part`), those chapters will be numbered "1", "1", and "2". If division numbers are inherited, those numbers will be "I.1", "II.1", "II.2".

In the 1.x versions of these stylesheets, all of the aspects of numeration were controlled by three now obsolete parameters: `$component-numbers-inherit`, `$division-numbers-inherit`, and `$section-numbers-inherit`. In the 2.x stylesheets, the various aspects can be controlled independently and the result is much more consistent, if a bit more complicated.

The default numeration parameters are designed to cover the most common use cases and are specified with strings so that they're easy to control with parameters. Any numeration scheme can be implemented with a

### 3.5. Controlling numeration

customization layer, but hopefully that will be necessary only rarely and in uncommon cases.

To simplify the problem, we divide the DocBook elements into six categories:  
sets

The `set` is the only member of this category.

books

The `book` is the only member of this category.

divisions

The divisions elements are `part` and `reference`.

components

The component elements are `acknowledgements`, `appendix`, `article`,  
`bibliography`, `chapter`, `colophon`, `dedication`, `glossary`, `index`, `partintro`,  
`preface`, `refentry`, and `setindex`.

sections

The section elements are `section`, `sect1`, `sect2`, `sect3`, `sect4`, `sect5`,  
`simplesect`. The `refentry` section elements are not included because they  
are not typically numbered.

formal objects

The formal objects are `figure`, `table`, `example`, `equation`, `formalgroup`,  
`procedure`.

There's a bit of complexity here. A `formalgroup` that contains figures  
counts as a `figure`, a `formalgroup` that contains tables counts as a `table`,  
etc. An `equation` or `procedure` only counts as a formal object if it has a  
title.

Six parameters control where numbering starts (or restarts):

`$sets-number-from`, `$books-number-from`, `$divisions-number-from`,  
`$components-number-from`, `$sections-number-from`, and `$formal-objects-number-from`. In  
each case, the value of the parameter must be the name of one of the  
categories. Sets and books can only number from sets, divisions can number  
from sets or books, components can number from sets, books, or divisions,  
etc. It is also possible to specify the value `root` to indicate that elements in the  
relevant category are numbered sequentially through the whole document.

To assure consistency, “numbering from” resets when the specified category or one of its ancestors is encountered. In other words, if you’re formatting a set of books and numbering components from divisions, the numbering resets when a new division, book, or set begins.

Six parameters control how numbers are inherited: `$sets-inherit-from`, `$books-inherit-from`, `$divisions-inherit-from`, `$components-inherit-from`, `$sections-inherit-from`, and `$formal-objects-inherit-from`. Like the “number from” parameters, each parameter takes the value of the categories above it. In this case, however, you can specify more than one category.

For example, the default value for formal objects is to inherit from “component section”. That means that the first figure in chapter 2 will be labeled “2.1” and the first figure in the first section in chapter 2 will be labeled “2.1.1”, etc. This most closely reproduces the numbering from the 1.x stylesheets.

## 3.6. Creating something completely different

Your input documents go through several pre-processing steps before they are rendered into HTML. If you want to produce completely different outputs, the place to start is with root template in the `m:docbook` mode.

Consider, for example (<https://github.com/docbook/xslTNG/issues/84>), the task of creating a JSON version of the Table of Contents. In principle, you could write your own stylesheet to do this, but leveraging the *DocBook xslTNG Stylesheets* means you can make use of functions like `f:generate-id()` to create links.

To produce completely different results, override the root template in the `m:docbook` mode:

```
1 <xsl:template match="/" mode="m:docbook">
  <xsl:document>
    <!-- your processing here -->
  </xsl:document>
5 </xsl:template>
```

This template *must* return a document node.

Note that you can mix-and-match your processing with default processing by processing DocBook elements in the `m:docbook` mode.

### 3.6. Creating something completely different

Here is a simple example of a stylesheet that produces a JSON version of the Table of Contents for a DocBook document:

Chapter 3. Customizing the stylesheets

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3     xmlns:db="http://docbook.org/ns/docbook"
4     xmlns:f="http://docbook.org/ns/docbook/functions"
5     xmlns:m="http://docbook.org/ns/docbook/modes"
6     xmlns:t="http://docbook.org/ns/docbook/templates"
7     xmlns:xs="http://www.w3.org/2001/XMLSchema"
8     xmlns="http://www.w3.org/1999/xhtml"
9     exclude-result-prefixes="db f m t xs"
10    version="3.0">
11
12    <!-- This href has to point to your local copy
13        of the stylesheets. --&gt;
14    &lt;xsl:import href="docbook/xslt/docbook.xsl"/&gt;
15
16    &lt;xsl:output method="text"/&gt;
17
18    <!-- Suppress xs/TNG's default HTML output; note that this template
19        must return a document node. --&gt;
20    &lt;xsl:template match="/" mode="m:docbook"&gt;
21        &lt;xsl:document&gt;
22            &lt;xsl:apply-templates select=".." mode="TOC"/&gt;
23        &lt;/xsl:document&gt;
24    &lt;/xsl:template&gt;
25
26    <!-- The templates below generate a simple JSON ToC. --&gt;
27
28    &lt;xsl:template match="/" mode="TOC"&gt;
29        {"toc": [
30            &lt;xsl:apply-templates mode="TOC"/&gt;
31        ]}
32    &lt;/xsl:template&gt;
33
34
35    &lt;xsl:template match="db:part|db:article|db:section|db:chapter" mode="TOC"
36        expand-text="yes"&gt;
37        &lt;xsl:if test="preceding-sibling::db:part
38            | preceding-sibling::db:article
39            | preceding-sibling::db:section
40            | preceding-sibling::db:chapter"&gt;
41            &lt;xsl:choose&gt;
42                &lt;xsl:when test="preceding-sibling::db:part"&gt;
43                    &lt;xsl:copy/&gt;
44                &lt;xsl:otherwise&gt;
45                    &lt;xsl:copy/&gt;
46                &lt;/xsl:otherwise&gt;
47            &lt;/xsl:choose&gt;
48        &lt;/xsl:if&gt;
49    &lt;/xsl:template&gt;</pre>
```

### 3.6. Creating something completely different

**Note**



This example is meant as a starting point; it's not robust as it only handles a few of the possible elements that might appear in a Table of Contents.

When processing documents this way, be aware that you are transforming the pre-processed, normalized versions of your input documents. For example, whether or not you put `info` wrappers around the titles of your sections, in the pre-processed input, `title`s *always* appear inside `info` wrappers. This normalization greatly simplifies processing in many places.



# Chapter 4. Localization

The *DocBook xslTNG* stylesheets support localization in more than 70 languages. At the time of this writing: Afrikaans, Albanian, Amharic, Arabic, Assamese, Asturian, Azerbaijani, Bangla, Basque, Bosnian, Bulgarian, Catalan, Chinese, Chinese (Taiwan), Chinese Simplified, Croatian, Czech, Danish, Dutch, English, Esperanto, Estonian, Finnish, French, Galician, Georgian, German, Greek, Gujarati, Hebrew, Hindi, Hungarian, Icelandic, Indian Bangla, Indonesian, Irish, Italian, Japanese, Kannada, Kirghiz, Korean, Latin, Latvian, Lithuanian, Low German, Malayalam, Marathi, Mongolian, Northern Sami, Norwegian Bokmål, Norwegian Nynorsk, Oriya, Polish, Portuguese, Portuguese (Brazil), Punjabi, Romanian, Russian, Serbian in Cyrillic script, Serbian in Latin script, Slovak, Slovenian, Spanish, Swedish, Tagalog, Tamil, Telugu, Thai, Turkish, Ukrainian, Urdu, Vietnamese, Welsh, and Xhosa.

## 4.1. Background

Near the end of the previous millennium, I was working on the DSSSL ([https://en.wikipedia.org/wiki/Document\\_Style\\_Semantics\\_and\\_Specification\\_Language](https://en.wikipedia.org/wiki/Document_Style_Semantics_and_Specification_Language)) stylesheets for DocBook. They were popular enough that users of languages other than English wanted to use them.

I invented a mechanism for doing simple localization so that the word “Chapter” in “Chapter 5” would, for example, be spelled “Chapitre” if the book was in French, and “Розділ” if it was in Ukrainian. What started as a simple word substitution system grew a few macro facilities and became a little more sophisticated<sup>1</sup>.

Over time, with the aid of dozens of volunteers around the world who contributed files for their languages, the DocBook stylesheets developed localization capabilities that were for the most part good enough.

Fast forward a few years and those language-specific localization files, and some of those mechanisms, were ported to the XSLT 1.0 stylesheets for DocBook.

Fast forward another decade and those XSLT 1.0 localization files and some of the mechanisms were ported to the XSLT 2.0 stylesheets for DocBook.

---

<sup>1</sup> I’m not sure you’d pick a twenty-something, monolingual Anglophone American to do this work if you were planning ahead, but that’s what happened.

Fast forward the better part of *another* decade and those XSLT 2.0 localization files and some of the mechanisms were ported to the *DocBook xslTNG* stylesheets.

Well. Sort of. Initially, I tried to replace the complex system of templates with a model that took the text that had to be generated and decomposed it into logical parts. It worked fine for English and many other languages, but didn't account for the complexity of many others, such as Chinese.<sup>1</sup>

Starting in version 2.0.0, the xslTNG stylesheets have reverted back to a templating system. The localization files have been transformed a little bit to make some of the customization easier (I hope). They can't stray too far from the original designs because I must reuse the localization data I have. I don't want to devise a system that requires another army of volunteers to provide new localization data.

### 4.1.1. Consequences

One unfortunate consequence of this history is that there's some cruft in the localization files. There are mappings and possibly templates that aren't actually used. Or, at least, they're not used in the standard DocBook stylesheet. They might be used in customization layers.

I made a few attempts to trim out cruft, but found all of the results unsatisfying. So, at least for the moment, I've left it in place. Like everything on earth, it's mostly harmless ([https://en.wikipedia.org/wiki/Mostly\\_Harmless](https://en.wikipedia.org/wiki/Mostly_Harmless)).

## 4.2. Overview

In this context, localization mostly refers to “generated text”, words and symbols that appear in a published DocBook document that aren't present in the original XML. Consider *Figure 4.1, “Sample book source”*.

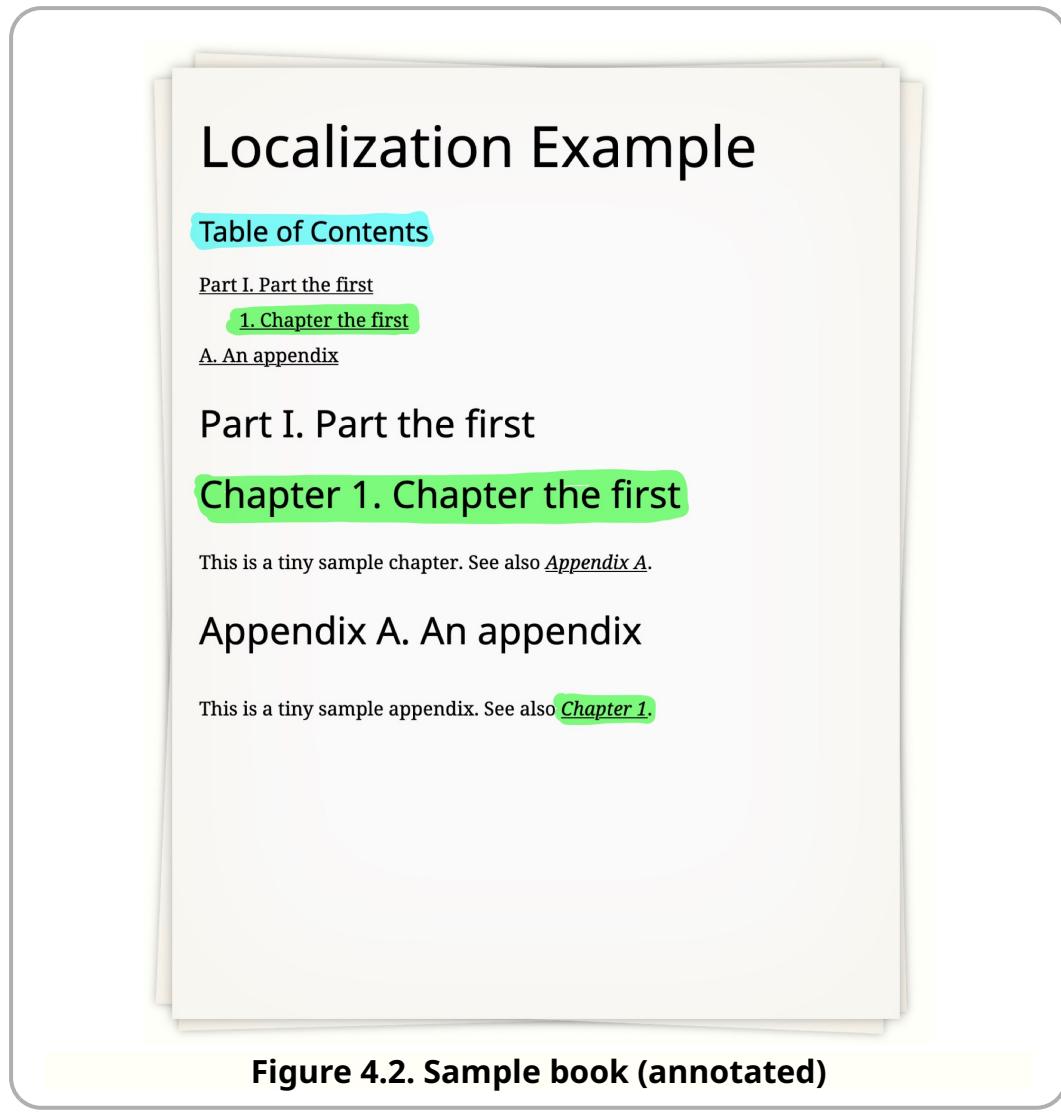
---

<sup>1</sup> Turns out a fifty-something, monolingual Anglophone American isn't much of an improvement, really.

```
1 <book xmlns="http://docbook.org/ns/docbook"
      version="5.0" xml:lang="en">
2   <info>
3     <title>Localization Example</title>
4   </info>
5   <part>
6     <title>Part the first</title>
7     <chapter xml:id="chap">
8       <title>Chapter the first</title>
9       <para>This is a tiny sample chapter.
10      See also <xref linkend="app"/>.</para>
11    </chapter>
12    </part>
13    <appendix xml:id="app">
14      <title>An appendix</title>
15      <para>This is a tiny sample appendix.
16      See also <xref linkend="chap"/>.</para>
17    </appendix>
18  </book>
```

**Figure 4.1. Sample book source**

It might be published as shown in *Figure 4.2, “Sample book (annotated)”*. Here we can see examples of several different kinds of generated text.



**Figure 4.2. Sample book (annotated)**

The title “Table of Contents” is entirely generated; it appears nowhere in the XML. The chapter title appears in the text, but it’s labeled “1.” in the list of titles, “Chapter 1.” in the chapter itself, and “Chapter 1” (without the title) in the cross reference.

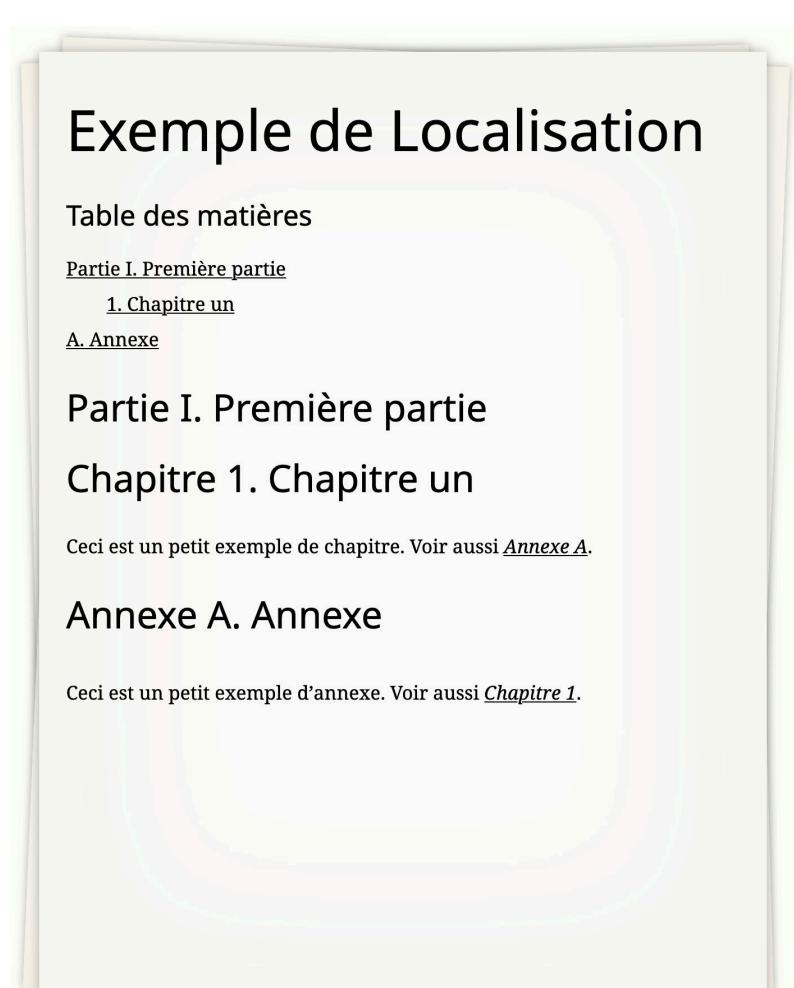
Now consider a French version of the document in *Figure 4.3, “Sample book source (French)”*.

## 4.2. Overview

```
1 <book xmlns="http://docbook.org/ns/docbook"
      version="5.0" xml:lang="fr">
2   <info>
3     <title>Exemple de Localisation</title>
4   </info>
5   <part>
6     <title>Première partie</title>
7     <chapter xml:id="chap">
8       <title>Chapitre un</title>
9       <para>Ceci est un petit exemple de chapitre.
10      Voir aussi <xref linkend="app"/>.</para>
11    </chapter>
12    </part>
13    <appendix xml:id="app">
14      <title>Annexe</title>
15      <para>Ceci est un petit exemple d'annexe.
16      Voir aussi <xref linkend="chap"/>.</para>
17    </appendix>
18  </book>
```

**Figure 4.3. Sample book source (French)**

In this case, the published version will have different localization, as shown in *Figure 4.4, “Sample book (French)”*.



**Figure 4.4. Sample book (French)**

The question is, how is this accomplished? The answer, I'm afraid, is not simple.

It begins with a localization file.

### 4.3. Localization files

The localization files are in `src/main/locale` in the repository. The localization file is designed to be simple enough to edit by hand. The stylesheets use compiled versions created by processing the input locale with `src/main/xslt/modules/xform-locale.xsl` to produce the files in `xslt/locale` in the distribution.

A locale begins by defining the language it supports and providing an English language name for it. The `language` attribute identifies the language (in the same terms as `xml:lang`) to which this localization applies.

### 4.3.1. Mappings

That's followed by metadata about the file (authors, etc.), then mappings, groups, lists, and letters as shown in *Figure 4.5, “Example locale file (excerpted)”*. We'll consider each section in detail below.

```
1 <locale xmlns="http://docbook.org/ns/docbook/I10n/source"
      xmlns:db="http://docbook.org/ns/docbook"
      language="en" english-language-name="English">
  5   <info>
    ...
  10  </info>
  15  <mappings>
    <gentext key="above">above</gentext>
    ...
  20  </mappings>
  25  <group name="title">
    <template match="self::db:chapter">{chapter} %l%.%c</template>
    ...
  30  </group>
  35  <list name="_default">
    <items>%c</items>
    <items>%c {and} %c</items>
    <items>%c<repeat>, %c</repeat>, {and} %c</items>
  40  </list>
  45  <letters>
    ...
  50  </letters>
</locale>
```

**Figure 4.5. Example locale file (excerpted)**

### 4.3.1. Mappings

The mappings section is a simple list of key/value pairs. Each `gentext` element defines a key and its replacement.

```

1 <mappings>
  <gentext key="above">above</gentext>
  <gentext key="abstract">Abstract</gentext>
  ...
5  <gentext key="xrefto">xref to</gentext>
</mappings>

```

These mappings serve two purposes. For many languages, a lot of the work of defining a new localization is just updating these mappings. For a stylesheet customization layer, it provides a mechanism for remapping on an ad hoc basis.

In a localization template, any key entered in curly braces will be replaced by the mapping. In other words, for the example above, `{abstract}` will be replaced by the word “Abstract”. This mapping is done when the document is transformed, not when the localization file is compiled.

### 4.3.2. Group

Groups are the primary templating system. In a context where generated text is required, a group is selected and within that group, a template is selected. The template is selected by evaluating the expression in the `match` attribute with the current node as the context item. A template with the match expression `true()` will always succeed; it is used as a fallback.

The `title-numbered` group determines how titles are formatted if they are numbered, (there's also `title-unnumbered` when titles aren't being numbered):

```

1 <group name="title-numbered">
  <template match="self::db:section[ancestor::db:preface]">%c</template>
  ...
5  <template match="self::db:appendix">{Appendix} %l%.%c</template>
  ...
  <template match="self::db:warning">%c</template>
  <template match="true()">%l%.%c</template>
</group>

```

(Note that not all titles *are* numbered, this is just the group that's used if they could be. See `$division-numbers`, `$component-numbers`, and `$section-numbers`.)

### 4.3.3. List

Within a template, two kinds of substitution are performed: names in curly braces are replaced by the corresponding mapping and %-letter values are substituted as follows:

**Table 4.1. Template %-letter substitutions**

#### %-letter Substitution

%c	The content (for example, the text of the title)
%l	The label (for example, “Chapter 1” or “see also”)
%p	The page number (not yet implemented)
%o	The olink title (not yet implemented)
%.	The separator (often, “.”)
%%	A literal % character

If the `title` group is being used to generate text for the chapter from our example document:

- The `chapter` context is used to select the template (`{chapter} %l%.%c`).
- The string `{chapter}` is replaced by the mapping for `chapter`, which is “Chapter” in English.
- The label `%l` is “1” because this is the first chapter. (In fact, constructing the label uses templates from the localization file as well.)
- The separator `%.` is “.”. (Like the label, this is also constructed from a separate query to the localization file.)
- And the content `%c` is “Chapter the first”. (There’s no markup in this title, but if there was, it would be retained. The content is a list of items, not a string.)
- Literal text, such as the non-breakable space between “`{chapter}`” and “`%l`”, is retained verbatim.

## 4.3.3. List

List elements are used to format items that can be repeated (terms in a variable list, lists of authors, lists of “see also” terms, etc.). The list consists of a series of items. Within each item, one or more content replacements is specified with `%c`. The items must be arranged so that there’s a match for one, two, three, etc. items.

If an item contains a repeat, that repeat will be used for as many items as necessary to complete the list formatting. The default list format is:

```

1 <list name="_default">
  <items>%c</items>
  <items>%c {and} %c</items>
  <items>%c<repeat>, %c</repeat>, {and} %c</items>
5 </list>
```

Consider how a list of four authors in an `authorgroup` would be formatted. Call them A, B, C, D, for simplicity (and assume there's no list for “`authorgroup`”, so the “`_default`” will be used).

The first two items match one and two items, respectively. They aren't appropriate for a list of four items. The third item contains three items and a repeat, so that can be used for a list of four (or more) items.

The first `%c` is “A”. The second `%c` is in a repeat, followed by another `%c`. There are three elements left in the list at this point, so two will be used in the repeat and the last one will follow it.

The result will be `A, B, C, and D` where the word “and” was found by looking for the `and` key in the mappings.

### 4.3.4. Letters

The letters group is used to identify the lexical order and grouping of letters.

```

1 <letters>
  <l i="-1"/>
  <l i="0">Symbols</l>
  <l i="10">A</l>
5 <l i="10">a</l>
  ...
  <l i="20">B</l>
  <l i="20">b</l>
  ...
10 </letters>
```

All of the symbols with the same “i” value will be grouped together.

## 4.4. Customizing a localization

This mechanism dates from the days before XSLT supported language-specific collations. It is used in generated indexes, but perhaps it should simply be phased out.

# 4.4. Customizing a localization

For many users, the localizations provided are entirely sufficient. But if you want to change them, you have a few options.

## 4.4.1. Replacing entire localization files

If you want to replace an entire localization file (if, for example, you want to apply the same changes to a set of stylesheets), you can approach that as follows:

1. Copy the localization source files.
2. Update the ones you wish to change.
3. Compile them all with `src/main/xslt/modules/xform-locale.xsl` saving the output in a new location.
4. In your stylesheet, change the `$v:localization-base-uri` to point to the directory where the new locales reside. Those locale files will be used.

## 4.4.2. Overriding mappings, groups, etc.

If you only want to override a small number of localization features, it may be simpler to do so directly in your stylesheet. The variable `$v:custom-localizations` will be merged with the default localizations before transformation begins.

Suppose, for example, that you wanted:

- The table of contents title to simply be “Contents”,
- To omit the word “Appendix” from the appendix title, and
- To change the form of the cross reference to appendixes to read “App. A” instead of “Appendix A”.

The following customization would accomplish that:

```

1 <xsl:variable name="v:custom-localizations">
2   <locale xmlns="http://docbook.org/ns/docbook/l10n/source"
3     language="en"
4     english-language-name="English">
5   <mappings>
6     <gentext key="TableofContents">Contents</gentext>
7   </mappings>
8   <group name="title-numbered">
9     <template name="appendix">%l%.%c</template>
10  </group>
11  <group name="xref-number">
12    <template name="appendix">App. %l</template>
13  </group>
14  </locale>
15 </xsl:variable>

```

Note that it defines (a portion of) a locale source file for the language `en`. These changes only apply to that locale.

This fragment replaces the mapping for `TableofContents` and the templates for numbered titles and numbered cross references.

To update multiple languages, put additional `locale` elements in the variable as siblings.

Formatting our example document above now produces:

#### 4.4.3. Changing the group

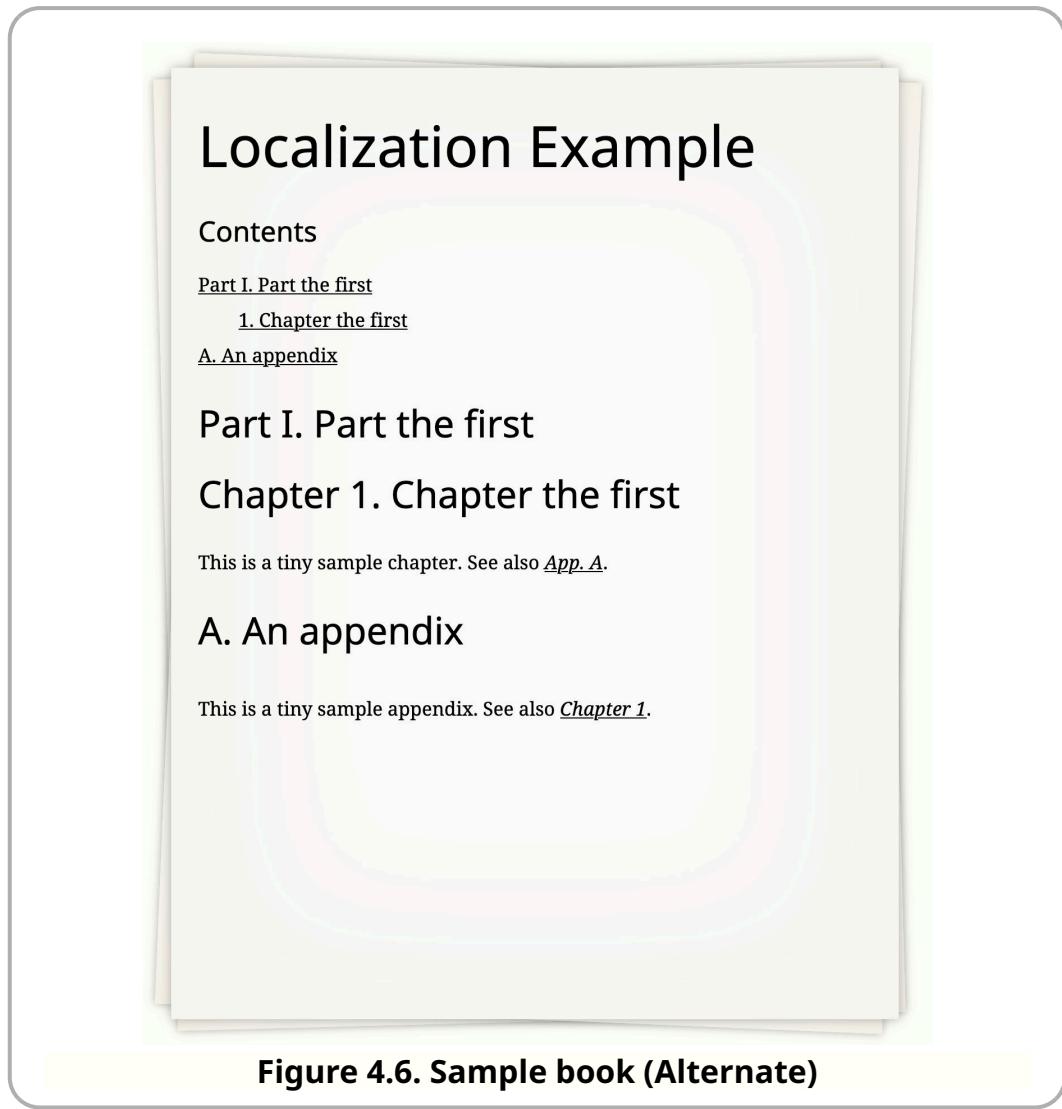


Figure 4.6. Sample book (Alternate)

#### 4.4.3. Changing the group

Sometimes, rather than change a template, you want to change which group of templates is used. This is controlled by two variables: `$v:user-title-groups` and `$v:user-xref-groups`.

##### 4.4.3.1. Changing the title group

The `$v:user-title-groups` element consists of a list of `title` elements, each with an `xpath` attribute, a `group` attribute, and an optional `template` attribute.

Suppose the stylesheet is trying to generate a title for an element. It considers each `title` element in turn. The `xpath` expression is evaluated with the element as the context item. If the effective boolean value of the expression is `true()`, then that title is selected and templates from the corresponding `group` are used.

If a `template` attribute is present, a template with that name is used. Otherwise the local name of the element is used as the template name.

By default, sections in a preface are not numbered. That's because the default title groups include:

```
<title xpath="self::db:section[ancestor::db:preface]"  
      group="title-unnumbered"/>
```

If you add a title that matches sections in a preface to `$v:user-title-groups`, it will take precedence. For example:

```
<title xpath="self::db:section"  
      group="title-numbered"/>
```

Because all of the user groups are consulted first, it isn't necessary to include the predicate that limits this title to sections in a preface (although it wouldn't change the result if you did).

### 4.4.3.2. Changing the cross reference group

Cross references are processed just like titles, except that the `$v:user-xref-groups` element consists of a list of `crossref` elements.

The default for cross references to chapters and appendixes is “`xref-number-and-title`”, so you get things like “Chapter 1. The Chapter Title”. In order to get a different presentation in the localization example used in this chapter, the following localization is used:

```
<xsl:variable name="v:user-xref-groups" as="element()*">  
  <crossref xpath="self::db:chapter|self::db:appendix"  
            context="xref-number"/>  
</xsl:variable>
```

That's why the cross reference to the first chapter is just “Chapter 1”.

## 4.5. Caveats

There's currently little documentation to tell you which group or template to change. The names are supposed to be somewhat self explanatory (for speakers of English), but sometimes you just have to look in the stylesheet.

## 4.5. Caveats

The `formalgroup` element is unique in DocBook in that its label depends on what it contains. A `formalgroup` of `figure` elements is itself a “Figure” where a `formalgroup` of `example` elements is an “Example”. If you need to change it, you may have to create your own template for the `formalgroup` element in the `m:headline` mode. The default version is in `modules/titles.xsl`.



# Chapter 5. Implementation

This section sketches out some features of the implementation. It would probably be better to build an annotated Definitive Guide (<https://tdg.docbook.org/>) or something, but this will have to do for now.

## 5.1. Customizing chunking

Chunking is controlled by the `$chunk-include` and `$chunk-exclude` parameters. These parameters are both strings that must contain an XPath expression.

For each node in the document, the `$chunk-include` parameter is evaluated. If it does not return an empty sequence, the element is considered a chunking candidate. In this case, the `$chunk-exclude` parameter is evaluated. If the exclude expression *does* return an empty sequence, then the element identified becomes a chunk. (If the exclude expression returns a non-empty value, the element will not become a chunk.)

## 5.2. Lengths and units

Lengths appear in the context of images (width and height) and tables (column widths). Several different units of length are possible: absolute lengths (e.g., 3in), relative lengths (e.g., 3\*), and percentages (e.g., 25%). In some contexts, these can be combined: a column width of “3\*+0.5in” should have a width equal to 3 times the relative width plus  $\frac{1}{2}$  inch.

In practice, some of the more complicated forms in *TR 9502:1995* have no direct mapping to the units available in HTML and CSS. The stylesheets attempt to specify a mapping that’s close. Broadly, they take the nominal width of the table (`$nominal-page-width`, subtract out the fixed widths, divide up the remaining widths proportionally among the relative widths, and compute final widths. The final widths can be expressed either in absolute terms or as percentages.

In handling the width and height of images, the intrinsic width and height of the image in pixels are converted into lengths by dividing by `$pixels-per-inch`. Nominal widths are taken into consideration if necessary.

### Note



Determining the intrinsic size of an image depends on an extension function. See [Section 2.5, “Extension functions”](#). Many bitmap image formats are supported. The bounding box of EPS images is used, if it’s present. The intrinsic size of SVG images is not available.

The list of recognized units (in, cm, etc.) are taken from [\\$v:unit-scale](#).

## 5.3. Verbatim styles

There are four verbatim styles: [lines](#), [table](#), [plain](#), and [raw](#).

[lines](#)

In the lines style, each line of the verbatim environment is marked up individually. In this style, lines can be numbered and callouts can be inserted.

[table](#)

In the table style, each line of the verbatim environment is marked up individually, very much like the lines style. In this style, lines can be numbered and callouts can be inserted. It differs from the lines style in that the whole thing is wrapped in a table.

The table has one row and two columns. The line numbers appear in the first column, the lines in the second. This format was added in order to improve the display in user agents that don’t support CSS. Ironically, in the course of adding this style, a number of changes were made to the way line numbers are formatted in the lines style making it largely, perhaps entirely, unnecessary.

[plain](#)

In the plain style, callouts can be inserted, but additional markup is not added (except for the callouts). Consequently, it isn’t possible to do line numbering or syntax highlighting. (It may be possible to provide these features with JavaScript libraries in the browser, however.)

### 5.3.1. Line numbers

raw

In the raw style, no changes are made to the verbatim content. It's output as it appears. Inline markup that it contains, `emphasis` or other elements, will be processed, but you cannot add line numbers, callouts, or syntax highlighting.

Consult *I. Parameter reference* for a variety of parameters that control aspects of verbatim processing.

## 5.3.1. Line numbers

In the `lines` and `table` styles, line numbers may be added to the beginning of some (or all) lines. Prior to version 1.10.0, the stylesheets inserted the numbers without any padding:

```
<span class="line">
  <span class="ln">5</span>
  <span class="ld">The line of text</span>
</span>
```

(The newlines and indentation in these examples are for clarity. In practice, these are inside a `pre` where every space counts and they're all run together with line breaks only occurring between lines.)

In a graphical browser with CSS support, this looked fine. But without CSS, the line numbers and the text that followed them could flow together and the alignment of the numbers was unclear.

Starting in version 1.10.0, the stylesheets insert padding spaces before each number so that they will all be aligned. If the largest line number is three digits long, every number smaller than 100 will be padded to a width of three characters. A single space is added after the number to separate it from the text that follows. An additional separator may also be inserted, as shown here.

```
<span class="line">
  <span class="ln"> 5 <span class="nsep">|</span></span>
  <span class="ld">The line of text</span>
</span>
```

These changes have no visible effect when CSS is used to style the verbatim environment. But without CSS, the numbers are aligned and separated from

the text that follows. The `$verbatim-number-separator` is generally suppressed by CSS, but is visible in text browsers.

## 5.4. Processing mediaobjects

Starting in version 1.11.0, the way media objects are processed has been refactored. This is designed to support fallback at both the object level (`imageobject`, `audioobject`, `videoobject`, `textobject`, and `imageobjectco`) and at the data level (`imagedata`, `audiodata`, `videodata`, and `textdata` within the objects).

Each data element and object element is processed in the `m:mediaobject-info` mode. This returns a map for each object that contains an array of maps, one for each data element:

**Table 5.1. The object map**

Key	Value
<code>content-types</code>	An array of the distinct content types in the data elements
<code>datas</code>	An array of data maps
<code>extensions</code>	An array of the distinct extensions in the data elements
<code>node</code>	The media object node

Each data map has the following structure:

**Table 5.2. The data map**

Key	Value
<code>align</code>	The alignment of the data (if specified)
<code>content-type</code>	The computed content type for the data
<code>contentheight</code>	The content height of the data (if specified)
<code>contentwidth</code>	The content width of the data (if specified)
<code>extension</code>	The extension of the data file (if there is one)
<code>fileref</code>	The original <code>fileref</code> attribute value
<code>height</code>	The height of the data (if specified)
<code>href</code>	The computed <code>href</code> value for the HTML element; this takes account of the <code>\$mediaobject-input-base-uri</code> and <code>\$mediaobject-output-base-uri</code> .
<code>node</code>	The data element
<code>params</code>	Any <code>multimediacparams</code> associated with the data element

## 5.4. Processing mediaobjects

Key	Value
properties	The properties of the element (as returned by the extension funtions; this can include EXIF data and metadata)
scale	The scaling factor (if there is one) or 1.0
scalefit	True if the image should be scaled to fit (implicitly or explicitly)
uri	The computed absolute URI of the input data
valign	The vertical alignment of the data (if specified)
width	The width of the data (if specifieid) <sup>1</sup>

The `uri` and `href` properties are computed by processing the data elements in the `m:mediaobject-uris` mode.

Armed with information about the objects and the data associated with them, the stylesheets proceed to choose an object and then process it. Each object is considered in turn, if any of the data elements it contains were excluded, then it is rejected. The first object where all of the elements are acceptable is selected.

Consider this example:

```
1 <mediaobject>
  <imageobject>
    <imagedata fileref="image.svg"/>
    <imagedata fileref="image.eps"/>
  5  <imagedata fileref="image.png" width="4in"/>
  </imageobject>
  <imageobject>
    <imagedata fileref="image.svg"/>
    <imagedata fileref="image.png" width="40em"/>
  10 </imageobject>
  </mediaobject>
```

If this is being processed for online presentation, the default value of `$mediaobject-exclude-extensions` will exclude the EPS file. Because one of its data elements was excluded, the processor will choose the object containing only the SVG and PNG images for online presentation.

---

<sup>1</sup> DocBook uses “depth” instead of “height”, but we convert it to height for consistency with most other systems

Once an object is selected, an appropriate wrapper is created and all of the alternatives are placed within it. So the example above will result in `picture` element containing a `source` for the SVG image and an `img` for the fallback PNG.

### Note



Consistent with HTML, only the size, scaling, and alignment attributes of the *last* alternative data element are considered! These apply irrespective of which alternative is selected.

### 5.4.1. Mediaobject URIs

Media object URIs are tricky to handle. It's most convenient if the URIs in the source documents point to the actual media files. This allows extensions, like the image properties extension function, to access the files. At the same time, the references generated in the HTML have to point to the locations where they will be published.

In previous versions, the stylesheets attempted (broadly) to use the relative difference between the input and output base URIs to work out the correct relative URIs for media. That imposed restrictions on the authoring environment that weren't always easy to work with. Starting in version 2.0.6, the mechanisms for finding sources and producing references in the output has changed. Three parameters are used:

`$mediaobject-input-base-uri`

If the `$mediaobject-input-base-uri` is empty (the default), then URIs in the source document are assumed to be relative to the base URI on which they occur. This is the usual case if you mix XML and media into the same directory structure on the filesystem.

If the `$mediaobject-input-base-uri` is not empty, it is used to resolve all media URIs. If it's initialized with a relative URI, that URI will be made absolute against the base URI if the input document.

`$mediaobject-output-base-uri`

If the `$mediaobject-output-base-uri` is empty (the default), then URIs in the output are treated as parallel to the URIs in the input. If the reference `../image.png` works in the source document, it's assumed that will also work in the output document.

## 5.5. Templates

If the `$mediaobject-input-base-uri` is not empty, it is the base URI used for media objects. If this is a relative URI, it is taken to be relative to the root of the output hierarchy.

Suppose the output base URI is `https://images.example.com/`, then a reference to `image.png` will appear as `https://images.example.com/image.png` in the output.

If the output base URI is `media/`, then a reference to `image.png` will appear as `media/image.png` in the output. If the document is chunked, the paths back to the output directory are relative. In other words, if the reference to `image.png` appears in a chunk that will be located at `back/appendix.html`, then the media URI will be `../media/image.png`.

### `$mediaobject-output-paths`

This parameter controls whether the relative paths in the input URIs apply to the output URIs as well. If the parameter is true, the output base URI is `media/`, and the input URI is `path/to/image.png`, then the output reference will be to `media/path/to/image.png`. If it's false, then the output reference will be to `media/image.png`.

For a further discussion of the options, see *Section 3.4, “Managing media”*.

#### Important



The stylesheets are not responsible for actually copying the media files into the correct locations in the output. The stylesheets only generate the HTML files and the references. You must copy the images and other media with some other process.

## 5.5. Templates

It's difficult to make title pages for documents easy to customize. There is *a lot* of variation between documents and the styles can have very precise design constraints. At the end of the day, if you need complete control, you can define a template that matches the element in the `m:generate-titlepage` mode and generate all of the markup you wish.

The default title page handling attempts to make some declarative customization possible by using templates. A typical header template looks like this:

```

1 <db:section>
  <header>
    <tmp:apply-templates select="db:title">
      <h2><tmp:content/></h2>
    </tmp:apply-templates>
    <tmp:apply-templates select="db:subtitle">
      <h3><tmp:content/></h3>
    </tmp:apply-templates>
  </header>
10 </db:section>

```

Any HTML element in the template will be copied to the output. The semantics of a “template apply templates” element (`tmp:apply-templates`) is that it runs the ordinary `xsl:apply-templates` instruction on the elements that match its select expression. If they result is the empty sequence (e.g. if there is no `subtitle`), nothing is output. If there is a result, the content of the `tmp:apply-templates` element is processed. Anywhere that `tmp:content` appears, the result of applying templates will be output.

In this example, if the title is “H<sub>2</sub>O” and there is no subtitle, the resulting HTML title page will be:

```

<header>
  <h2>H<sub>2</sub>O</h2>
</header>

```

## 5.6. Annotations

The stylesheets fully support annotations, including a number of presentation styles enabled by JavaScript in the browser. They also support an extension of the documented semantics of `annotation`.

Annotations are applied to elements with links. Either the element must point to its annotations (with an `annotations` attribute) or the annotations must point to the elements they annotate (with an `annotates` attribute). These are documented as ID/IDREF links but they are not `IDREFS` attributes because annotations may be stored separately.

## 5.6. Annotations

Starting in version 1.5.1, the *DocBook xslTNG Stylesheets*<sup>1</sup> support a non-standard extension: if you place the string `xpath:` in the `annotate` attribute of an `annotation`, then the rest of the attribute is assumed to contain an XPath expression that points to the element(s) to which the annotation applies. (If you want to put IDREF values before the `xpath:` token, that's fine, but you can't put them after; the expression continues to the end of the attribute value.)

Suppose, for example, that you wanted to annotate the stylesheet title in the previous paragraph. The standard mechanisms would require that you either put an `xml:id` attribute on the element or point to the annotation from the element. With the XPath extension, you can do this:

```
1 <annotation
  annotates="xpath:preceding-sibling::db:para/db:citetitle"
  xmlns:db="http://docbook.org/ns/docbook">
<para>This annotation applies to the stylesheet title.
5 For a discussion of this annotation, see the
  following paragraphs.</para>
</annotation>
```

When the XPath expression is evaluated, the `annotation` element is the context item. Often, this means that you'll want to start the expression with `id()` or `/`.

The namespace context for the expression is also the `annotation` element, that's why I've added the DocBook namespace binding for the `db` prefix. In practice, if you're doing this on several annotations, you can just put all the namespace bindings on a common ancestor. All of the bindings *in scope* on the `annotation` element are available in the expression.

Caveats:

1. There's no way to have multiple XPath expressions. You can't put “`xpath:`” in there twice. If you want an annotation to apply to multiple elements, you'll have to construct a single expression that selects all the elements, or duplicate the annotation, or use ID/IDREF links.

If this turns out to be a serious limitation in practice, additional syntax could be added to support multiple expressions, but it doesn't *seem* necessary.

---

<sup>1</sup> This annotation applies to the stylesheet title. For a discussion of this annotation, see the following paragraphs.

2. You can only select elements. There's no way to select the third word in a particular paragraph, for example, unless it already has some markup around it. There's also no way to select a comment or a processing instruction.

The placement of the annotation marker (“ $\oplus$ ” by default) can also be controlled globally and on individual annotations. The `$annotation-placement` parameter provides global control. To specify the position for an individual annotation, put the token “`before`” or “`after`” in the `role` attribute on the `annotation`.

## 5.7. The pre- and post-processing pipeline

Processing a DocBook document is a multi-stage process. The original document is transformed several times before converting it to HTML. The standard transformations are:

1. Adjust the logical structure. Adds an XML base attribute to the root of the document and converts media object `entityref` attributes into `fileref` attributes.
2. Perform XInclude processing. Only occurs if the appropriate extension function is available and if the document contains `XInclude` element.
3. Convert DocBook 4.x to DocBook 5.x. Only occurs if the root element is not in a namespace.
4. Perform transclusion.
5. Perform profiling.
6. Normalize the content. This removes a lot of variation that's allowed for authoring. For example, authors aren't required to use an `info` element if a formal object has only a title. This process adds the `info` element if it's missing.
7. Resolve annotations.
8. Process XLink link bases.
9. Validate. Only occurs if the appropriate extension function is available and the stylesheet specifies a `$relax-ng-grammar`.
10. Process Oxygen change markup. Only occurs if `$oxy-markup` is true and the document contains Oxygen change markup processing instructions.

## 5.7. The pre- and post-processing pipeline

A customization can introduce transformations to the original document: before the standard transformations by specifying them in `$transform-original`; after the standard transformations but before the transformation to HTML by specifying them in `$transform-before`; or after the HTML transformation by specifying them in `$transform-after`. (If you need to insert a transformation in the middle of the standard transformations, you'll have to update the `$v:standard-transforms` variable.)

### Note



Transformations in `($v:transform-after)` will be processing the *HTML* result of applying the “main” DocBook transformation.

Each of the transformation variables holds a list of transforms that will be applied in the order specified. Each member of the list can be a map or a string. If a string is provided, it's the equivalent of providing this map:

```
map {  
    'stylesheet-location': $the-string  
}
```

The map can have several keys:

**Table 5.3. The transformation map**

Key	Value
<code>stylesheet-location</code>	The location of the XSLT stylesheet that performs this transformation. This key is required.
<code>extra-params</code>	A map of QName/value pairs. These parameters will be made available to the transformation in addition to all of the standard parameters available to a standard DocBook stylesheet.
<code>functions</code>	A list of functions (expressed as EQNames). The transformation will only be run if every extension function listed is available.
<code>test</code>	An arbitrary XPath expression. The expression will be evaluated with the document as the context item. If it returns an effective boolean value of true, the transformation will be run.



# Chapter 6. Building the stylesheets

If you wish, you can also clone the distribution and build them yourself. The distribution is designed to be self contained. In a Unix or Mac environment, running:

```
./gradlew dist
```

will build the stylesheets. Building will:

1. Compile the stylesheets and run the unit tests. The compiled stylesheets will be available in `build/xslt`.

Running all the tests requires building the reference guide and a few other things. You can build the stylesheets without running any of the tests with the `makeXslt` task.

2. Compile the extension functions. The compiled extension functions will be available in `build/libs`. The `jar` task will compile the extensions without running the tests.
3. “Compile” the Python script that helps run the stylesheets. (It’s not really compiled, but several stylesheet-version-specific strings are interpolated.) The `copyBin` task will setup the Python script without running the tests.

## Important



In principle, it should be possible to build the stylesheets on Windows™. In practice, it doesn’t work. One would imagine that mapping path (“`:`” → `,`) and filename (“`\`” → `/`) separators and constructing URIs for paths (“`C:\Users\...`” → “`file:///C:/Users/...`”) would make it work. In the author’s experience, that is not the case. Pull requests are welcome. It may be more expedient to build in the Linux subsystem on Windows 10 or in a Docker container.

## 6.1. Prerequisites

In order to build the stylesheets, you must configure your system with several prerequisites:

- Gradle (<https://gradle.org/>). The stylesheet builds use the Gradle wrapper ([https://docs.gradle.org/current/userguide/gradle\\_wrapper.html](https://docs.gradle.org/current/userguide/gradle_wrapper.html)) to assure a consistent environment across systems, it'll be downloaded automatically the first time you build.
- A modern version of Java. (Java 1.8 or later, for example.)
- Python (<https://www.python.org/>) 3 and the click (<https://click.palletsprojects.com/en/7.x/>) module. The Pygmentize (<https://pygments.org/>) program is also required for syntax highlighting, though that's not technically a build requirement.

If you discover other prerequisites that have been overlooked, or have questions or suggestions about how best to manage them, please let us know.

### 6.1.1. Saxon EE

You don't need a Saxon EE license to build or run the stylesheets. If you change any of the transforms in `src/main/xslt/transforms`, you will need a Saxon EE license to recompile them as exported SEF files.

If you don't have an EE license, the build process will simply omit the SEF versions and the "source" versions, including any modifications that you've made, will be used.

That's the theory, anyway. I've tested it, but if you have any trouble, please open an issue.

## 6.2. Repository structure

The most significant parts of the repository hierarchy are:

```
src/main/xslt  
src/main/xslt/transforms  
src/main/xslt/modules
```

These are the sources for the stylesheets themselves. The `transform` subdirectory contains the preprocessing stylesheets that are run as separate

## 6.2. Repository structure

transforms. The `modules` directory doesn't have any special significance, it just makes the main entry points easier to find.

You cannot run the XSLT stylesheets directly from the source location. You must build them first with the `makeXslt` build target.

```
src/main/web  
src/main/web/resources/css  
src/main/web/resources/js
```

These are the CSS and JavaScript files needed for accurate rendering in the browser or formatter process.

```
src/main/locales/locale-10  
src/main/locales/locale
```

The `local-10` directory holds copies of the localization files from the XSLT 1.0 stylesheets. They're transformed into the `locale` versions by the build system. This whole area is one that needs work.

```
src/test/xspec  
src/test/generators  
src/test/resources/xml  
src/test/resources/expected
```

These are the testing resources. You cannot run the XSpec tests directly from `src/test/xspec`. The build system copies them into `build/xspec` along with a few XSpec tests generated automatically by the stylesheets in `generators`.

The library of DocBook documents that are used for testing is stored in the `xml` directory. The HTML files in `expected` correspond to the expected results. The expected results aren't usefully viewed in a browser. Only the `body` element and its descendants are stored in the expected results. This avoids a lot of noise in the `head`.

```
src/main/java
```

These are the sources for the extension functions.

```
src/guide  
src/website
```

These are the sources, resources, and ancillary files for the reference guide and the website.

### tools

The `tools` directory holds a number of stylesheets and other scripts used by the build process.

### lib

The `lib` directory holds third party jar files. This is also where you can put your Saxon PE or EE files if you have them.

### build

#### build/actual

#### build/xslt

The build system puts all of its temporary files under `build`. It's always safe to delete the entire directory and start over, though it will require internet access and it may take a while.

Test files that you format are published to `build/actual` and the images, CSS, and JavaScript resources are copied under there so that everything will look right in the browser. For security reasons, some JavaScript features may not work if you are looking at the documents from the filesystem. You can work around this by pointing a local web server at the `build` directory.

The built stylesheets are in `build/xslt`. You can run them directly from there.

## 6.3. Build tasks

The build system is *Gradle*. Gradle's processing model operates in several phases, this allows an initial configuration phase to construct build targets (called tasks) dynamically. The *DocBook xslTNG Stylesheets* build script uses this facility quite a bit.

In the discussion that follows, `testdoc` is the base name of a test document, that is, one of the example files from the `src/test/resources/xml` directory. For example, one of the test documents is `src/test/resources/xml/article.001.xml`. In the build targets that would be the “`testdoc`” `article.001`. In an analogous way, “`testset`” is the root name of an XSpec test file in `src/test/xspec`.

The most important tasks are:

## 6.4. Testing tasks

`makeXslt`

This task “compiles” the stylesheets into `build/xslt`.

`report`

This task runs all of the tests and generates a unified report in `build/report`. This is the default task.

`test`

Runs the test suite against the stylesheets.

`jar`

Compiles the extension functions and creates the jar file in `build/libs`.

`dist`

Builds an archive in `build/distributions` suitable for distribution.

`guide`

Builds the reference guide.

`website`

Builds the website.

`explorer`

Generates the XSLT Explorer (<https://xslt.xmlexplorer.com/>) report for the stylesheets in `build/explorer`.

`helloWorld`

A smoke test target. It just prints a message, but doing so will exercise a bunch of the build machinery in Gradle.

## 6.4. Testing tasks

The way tests are managed was completely refactored in version 2.0.13. The new system is largely automatic.

1. Create a new test document in `src/test/resources/xml`. For this example, we'll create `para.003.xml`, an example of a `sidebar` in a `para`.

## Chapter 6. Building the stylesheets

2. Use the build target `para.003.html` to format the document. This build target (and several others) exist automatically because the source file has been created.
3. Inspect the results (in `build/actual/para.003.html`) and adjust the stylesheets until you are satisfied that the results are correct. Each time you make a stylesheet change, you can re-run the `para.003.html` target to rebuild the output.
4. When you are satisfied, run `param.003.expected` to update the expected results.
5. Run the `report` target to make sure no other tests have broken as a result of your changes.

More generally, the following targets exist for any source document in `src/test/resources/xml`:

`testdoc.html`

Formats `testdoc` into HTML and stores the result in `build/actual` where it can be viewed in the browser with the appropriate CSS, JavaScript, images, etc.

`testdoc.expected`

Formats `testdoc` into HTML and stores the result in `src/test/resources/expected`. You should *only* run this task when you've made a change and determined that the new results are correct and should replace the previously expected results.

`testdoc.pdf.html`

Formats `testdoc` into HTML suitable for paged media and stores the result in `build/actual`.

`testdoc.pdf`

The `.pdf` target will generate the `.pdf.html` output and then attempt to transform it into PDF with either AntennaHouse or PrinceXML. Additional configuration may be necessary to get this to work.

`testdoc.chunk`

Formats `testdoc` into HTML with chunking turned on. The result is stored in `index.html` and other files in `build/actual`. No attempt is made to avoid having the output files from different documents overwrite each

## 6.5. Running XSpec

other. At the moment, none of the automated tests create chunked output.

`testset.xspec`

Runs the *testset* set of XSpec tests. The test sets that are available are the sets constructed from build environments and the sets created by hand (in `src/test/xspec`).a

## 6.5. Running XSpec

In order to get consistent results across different runs and in different environments, the XSpec tests run a driver stylesheet, `xspec-driver.xsl`.

Note also that the XSpec shell script is modified by the stylesheets.



# Appendix A. Changes in version 2.1.0

Version 2.x of the *DocBook xslTNG Stylesheets* include some substantial changes from version 1.x. If you use the stylesheets “out of the box” without any customization, you should have no trouble updating from version 1.x to version 2.1.0 (or later), but there will be differences in the HTML produced. If you have customization layers for 1.x, you may need to change them in order to get the results you expect from 2.x.

The most significant rewrite, and the change that actually motivated the decision to adopt some backwards incompatible changes, is in the area of localization. Version 1.x of the *xslTNG* stylesheets attempted to simplify (from the preceding XSLT 2.0 stylesheets) how generated text was constructed. The model adopted simply didn’t work for some languages. Trying to adapt the model to support these languages was going to result in something *even more complicated* than what had worked before, so the model has largely been reverted to what it was before.

The format of the localization data and the way that it’s used are the subject of *Chapter 4, Localization*. In broad terms, the new system should produce “the same results” as the old system for an equivalent localization, but there may be small changes in the way lists of titles and cross references are formatted.

The major changes are:

- Completely reworked how localization is handled. See *Chapter 4, Localization*. The localization file format has changed. Localization files are no longer simply transformed from the XSLT 1.0 stylesheet format.
- Completely reworked how numbering of elements is performed. See *Section 3.5, “Controlling numeration”*. It’s a bit more complicated now, but it’s also more flexible and better tested. The default numeration should be the same as before, although a few small changes may appear in places where (I think) the previous numeration was wrong.
- Completely refactored the way that media objects are located during the transformation and how the URIs for them are generated in the output. It is now possible to store the media objects in a location independent of the source files. Four separate source arrangements are represented in the test suite and each can be processed to produce several different outputs; see the `all_mo_tests` build target and its dependencies.

## Appendix A. Changes in version 2.1.0

- Refactored the templates for processing media objects to improve support for accessibility metadata and formatting copyright statements, legal notices, etc. See `m:mediaobject-start` and `m:mediaobject-end`.
- Fixed the bug where attributes on image, video, audio, and text objects in `mediaobject` and `inlinemediaobject` were not being preserved in the output. In particular, this meant that `role` attributes on those objects were not reflected in the HTML `class` attribute.
- Reworked audio and video fallback. Placing an HTML prose description in the `video` tag is not the correct approach in modern browsers. That prose is only rendered by browsers that don't understand the `video` tag *at all*. Apparently, it has to be handled by JavaScript. There's now a `$fallback-js` parameter.
- Support media objects that have no media (e.g., a media object that contains only inline text objects).
- Implemented an on-page table of contents feature.
- Switched to `section` elements (instead of `div` elements) for sections inside of `refentry` and a number of other places. This supports the new on-page table of contents feature and should have been done when the rest of the sectioning elements were converted.
- Reworked ToC handling. Added `$auto-toc` parameter. The placement of generated lists-of-titles can now be controlled with an empty `toc` element (or `db-toc` processing instruction).
- Added support for creating ToCs by hand. (Former handling for `tocdiv` and `tocentry` was just broken.)
- Placement of table titles, before or after the table, is now set with `$formal-object-title-placement`. For backwards compatibility, it defaults to 'before' for tables. Within a `formalgroup`, title placement is controlled by `$formalgroup-nested-object-title-placement`. In a related way, the placement of media object details can be set with `$mediaobject-details-placement`.
- Labels for formal objects may be different, but I think the previous formatting was actually in error.
- Changed `$mediaobject-accessibility` and `$table-accessibility` parameters into space-separated lists of strings.
- The parameters for the `t:top-nav` and `t:bottom-nav` templates have changed. They are now always called, even when chunking is not being

## 6.5. Running XSpec

performed. The new `$chunk` parameter indicates whether or not a chunk is being formatted.

- Moved the `footer` element from just after `main` to just inside it. This simplifies and improves CSS rendering and is arguably more correct.
- Added an `$unwrap-paragraphs` parameter. If this parameter is true, a single DocBook paragraph that contains block elements (for example, tables or figures) will be rendered as several HTML paragraphs with blocks between them. HTML doesn't allow blocks inside paragraphs.
- Made including the `docbook-print.css` file conditional on producing print output. Modern browsers attempt to parse the file, even when the link specifies that it's for print media, and produce a large number of spurious error messages.
- Deprecated the `m:html-body-script` mode. Changed the way the standard scripts are included; instead of putting them at the end of the `body` element, they're placed in the `head` but explicitly marked deferred.
- In some contexts, for example callouts, labels are now used for the links instead of titles.
- Removed up-arrow from the keys that `$chunk-nav` responds to. The up-arrow key is used by browser to move up the page and chunk navigation was interfering with that behavior. The `U` key will still go "up" in chunk navigation.
- Reworked the persistent ToC. It can now store the ToC in a separate file. Improved the error messages when the persistent ToC is unavailable.
- Changed `ext:cwd()` so that it always returns an absolute URI. This fixes a bug in chunk output base URI handling. Tidied up some code.
- Process unexpected elements in titlepage templates in the normal way; removed the warning message associated with them. Add it back by putting `templates` in `$debug`.
- Improved presentation of multiple `keycap` elements in a `keycombo`.
- Added table-of-contents to the linear flow of EPUBs. This fixes an epubcheck 3.3 error. Fixed the CSS for the ToC.
- Added support for ISO 690 bibliographies via a `$bibliography-style` parameter.
- Added a `$message-level` parameter to support suppressing some informational messages.

## Appendix A. Changes in version 2.1.0

- Removed `db-footnote` attributes from the output. Technically, attribute names that contain a hyphen are HTML5 extension attributes, but sometimes they make validation more difficult. Some extension attributes remain, because they're used by JavaScript in the presentation for example, but `db-footnote` isn't used that way.
- Fixed bug where `authorgroup` wasn't being processed in `biblioentry`.
- Fixed a bug where `orderedlist` numeration was not handled correctly in cross-references to list items.
- Fixed incorrect URIs for SVG draft overlay in print CSS.
- Fixed a bug where the `catalog.xml` file in the jar file was not at a location where the XML Resolver would find it.
- Updated the build system to use Pygments version 2.14.0. This effects the markup produced in some syntax highlighted listings. (You're free to use any version of Pygments that's convenient; this is just about conformance with the expected test results.)
- The way unit tests are managed has been completely rewritten. See *Section 6.4, “Testing tasks”*.

# Glossary

## Clark name

A Clark name is a way of unambiguously representing a qualified name (see *XML*) as a string. It consists of the namespace name in curly braces followed immediately by the local name. For example, the Clark name for the DocBook `para` element is: “`{http://docbook.org/ns/docbook}para`”.

## hamburger menu

The “hamburger menu” or “hamburger button ([https://en.wikipedia.org/wiki/Hamburger\\_button](https://en.wikipedia.org/wiki/Hamburger_button))” is the name for the “≡” icon often used to indicate a menu of choices.

## intrinsic size

The intrinsic size of an object is its actual size, measured in absolute units, usually pixels. The stylesheets rely on extension functions to obtain the intrinsic size of objects; if the extension functions aren’t available or the image format is not recognized, no intrinsic size will be known.

## is true

XPath defines a boolean type, `xs:boolean`, that is either true or false. But for the purpose of passing parameters to the stylesheet at runtime, whether it’s from the command line, from a configuration file, or through some other means, it’s often more convenient to pass string values. This saves the user from having to use whatever extra syntactic mechanisms would be required to identify the type of the variable.

To this end, many “boolean” parameters are actually strings. A string value `is true` if it consists of one of the strings “true”, “yes”, or “1”.

## object

In this guide, the term “object” is used to refer generally to any image, video, or audio media element. They have a width and a height but no useful internal structure from the perspective of the stylesheets.

## pseudo-attribute

Processing instructions (see *XML*) have no internal structure. The stylesheets interrogate the `db` processing instruction to determine formatting properties for a number of different elements.

## Glossary

In order to make it easy to consistently specify different properties, the value of the processing instruction is parsed as if it contained attributes. These *pseudo-attributes* must consist of a name, followed by “`=`”, followed by a quoted string.

## References

[AntennaHouse] *AH Formatter* (<https://www.antennahouse.com/>). Version 7.0.3.

[ISO 8601] *ISO 8601-1:2019 Date and time — Representations for information interchange — Part 1: Basic rules*. ISO (International Organization for Standardization). 2019-02.

[CSS] *What Is CSS?* (<https://www.w3.org/Style/CSS/>). A family of standards developed by the W3C.

[TR 9901:1999] *XML Exchange Table Model Document Type Definition* (<https://www.oasis-open.org/specs/tm9901.htm>). Organization for the Advancement of Structured Information Standards (OASIS) Technical Memorandum TR 9901:1999. Tables Technical Committee. Norman Walsh, editor.

[TR 9502:1995] *CALS Table Model Document Type Definition* (<https://www.oasis-open.org/specs/tm9502.html>). SGML Open Technical Memorandum TM 9502:1995. Table Interchange Subcommittee. Harvey Bingham, editor.

[Gradle] *Gradle* (<https://gradle.org/>).

[RFC 5147] *URI Fragment Identifiers for the text/plain Media Type* (<https://tools.ietf.org/html/rfc5147>). E. Wilde and M. Duerst, editors. Internet Engineering Task Force. 2008.

[MathJax] *MathJax* (<https://www.mathjax.org/>).

[OldMeasure] *The Old Measure: An Inquiry Into the Origins of the U.S. Customary System of Weights and Measures* (<https://www.amazon.com/Old-Measure-Inquiry-Customary-Measures/dp/0615376266>). 2010. Jon Bosak.

[Prince] *Prince* (<https://www.princexml.com/>). Version 13.5.

[Transclusion] *DocBook Transclusion* (<https://docbook.org/docs/transclusion/transclusion.html>). 2015. Jirka Koskek.

[Names] *Falsehoods Programmers Believe About Names* (<https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/>). 2010. Patrick McKenzie.

[XML] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, editors. *Extensible Markup Language (XML) 1.0 Fifth Edition* (<http://www.w3.org/TR/REC-xml>). World Wide Web Consortium, 2008.

## References

[XInclude] Jonathan Marsh, David Orchard, and Daniel Veillard, editors. *XML Inclusions (XInclude) Version 1.0 (Second Edition)* (<http://www.w3.org/TR/xinclude>). World Wide Web Consortium, 2006.

[Pygments] *Pygments: Python Syntax Highlighter* (<https://pygments.org/>). Version 2.6.1.

[highlight.js] *highlight.js* (<https://highlightjs.org/>). Version 10.1.2.

[Prism] *Prism* (<https://prismjs.com/>). Version 1.20.0.

[metadata-extractor] *metadata-extractor* (<https://drewnoakes.com/code/exif/>). Version 2.14.0.

[search] *Searching text/plain documents with a fragid* (<https://norman.walsh.name/2016/09/29/search>).

[XSLT 1.0] *XSL Transformations (XSLT) Version 1.0* (<https://www.w3.org/TR/1999/REC-xslt-19991116>). James Clark, editor. W3C Recommendation. 16 November 1999.

[XSLT 2.0] *XSL Transformations (XSLT) Version 2.0* (<http://www.w3.org/TR/xslt20>). Michael Kay, editor. W3C Recommendation. 23 January 2007.

[XSLT 3.0] *XSL Transformations (XSLT) Version 3.0* (<http://www.w3.org/TR/xslt-30>). Michael Kay, editor. W3C Recommendation. 8 June 2017.

[XSL FO] *Extensible Stylesheet Language (XSL) Version 1.1* (<https://www.w3.org/TR/xsl>). Anders Berglund, Editor. W3C Recommendation. 5 December 2006.

# Acknowledgements

I'd like to thank The Academy, the...no, wrong event.

DocBook has benefited immeasurably by the time and energy that innumerable people have poured into it. I thank them all.

This project has benefited from the generous support of Michael Kay and Saxonica for *Saxon EE* and Micheal Miller and Antenna House for *Antenna House Formatter*.

Bethan Tovey-Walsh designed the lovely xslTNG logo.



# Index

## D

db processing instruction, 99

## P

### param

- resource-base-uri, 20
- chunk, 27
- chunk-output-base-uri, 28
- chunk-include, 28, 75, 75
- chunk-exclude, 28, 75, 75
- persistent-toc, 28, 46
- chunk-nav, 28, 29, 97
- profile-os, 31, 32
- profile-arch, 32
- profile-audience, 32
- profile-condition, 32
- profile-conformance, 32
- profile-lang, 32
- profile-outputformat, 32
- profile-revision, 32
- profile-revisionflag, 32
- profile-role, 32
- profile-security, 32
- profile-userlevel, 32
- profile-vendor, 32
- profile-wordsize, 32
- profile-separator, 32, 32
- dynamic-profiles, 32
- verbatim-syntax-highlighter, 33
- persistent-toc-css, 34, 46
- persistent-toc-search, 34
- persistent-toc-filename, 34, 35, 35
- pagetoc-elements, 36
- pagetoc-dynamic, 36
- pagetoc-js, 37
- verbatim-syntax-highlight-css, 46
- use-docbook-css, 46
- user-css-links, 46
- sets-number-from, 53
- books-number-from, 53
- divisions-number-from, 53

- components-number-from, 53
- sections-number-from, 53
- formal-objects-number-from, 53
- sets-inherit-from, 54
- books-inherit-from, 54
- divisions-inherit-from, 54
- components-inherit-from, 54
- sections-inherit-from, 54
- formal-objects-inherit-from, 54
- division-numbers, 66
- component-numbers, 66
- section-numbers, 66
- nominal-page-width, 75
- pixels-per-inch, 75
- verbatim-number-separator, 78
- mediaobject-input-base-uri, 78, 80, 80, 80, 81
- mediaobject-output-base-uri, 78, 80, 80
- mediaobject-exclude-extensions, 79
- mediaobject-output-paths, 81
- annotation-placement, 84
- relax-ng-grammar, 84
- oxy-markup, 84
- transform-original, 85
- transform-before, 85
- transform-after, 85
- fallback-js, 96
- auto-toc, 96
- formal-object-title-placement, 96
- formalgroup-nested-object-title-placement, 96
- mediaobject-details-placement, 96
- mediaobject-accessibility, 96
- table-accessibility, 96
- unwrap-paragraphs, 97
- debug, 97
- bibliography-style, 97
- message-level, 97

## Index

### V

#### variable

v:localization-base-uri, 69  
v:custom-localizations, 69

v:user-title-groups, 71, 71, 72  
v:user-xref-groups, 71, 72  
v:unit-scale, 76  
v:standard-transforms, 85  
v:transform-after, 85

# Index by module

Many modules define related functions, templates, or variables.

## **VERSION.xsl**

\$v:VERSION, \$v:VERSION-ID

## **highlight.xsl**

, f:syntax-highlight(), f:syntax-highlight(), f:syntax-highlight()

## **main.xsl**

m:docbook, \$output-media

## **modules/admonitions.xsl**

m:docbook

## **modules/annotations.xsl**

m:annotation-content, m:docbook

## **modules/attributes.xsl**

m:attributes, m:docbook

## **modules/biblio690.xsl**

m:biblio690,

## **modules/bibliography.xsl**

m:biblioentry, m:bibliomixed, m:docbook

## **modules(blocks.xsl**

m:docbook, m:revhistory-list, m:revhistory-table, m:toc

## **modules/chunk-cleanup.xsl**

t:bottom-nav, m:chunk-cleanup, t:chunk-footnotes, m:chunk-title, f:chunk-title(), , m:footnote-number, , m:mediaobject-output-adjust, t:top-nav

# Part II. Reference



# I. Parameter reference

# \$additional-languages

Additional localization languages (beyond the default language)

Parameter:	{}additional-languages
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$additional-languages := ()
```

## Description

If the stylesheets need to generate text (the names of labels such as “Chapter”, for example), they attempt to do so in the same language as the document. This is determined by finding the nearest in-scope `xml:lang` attribute from the context where the generated text is required.

The stylesheets can generate text in more than 70 languages, but very few documents use more than a few. To avoid the overhead of loading a great many localizations that will never be used, the stylesheets don’t load them all.

The stylesheets always load the `$default-language` and will load any additional languages identified in `$additional-languages`, a space separated list of language identifiers.

If the `$additional-languages` is the empty sequence, the stylesheets attempt to determine which languages are used in the document and load the appropriate localizations.

# \$align-char-default, \$align-char-pad, \$align-char-width

Support “char” alignment on CALS table cells.

Parameter:	{align-char-default}
	{align-char-pad}
	{align-char-width}
Defined in:	param.xsl (3)
Used in:	param.xsl, modules/tablecals.xsl

## Synopsis

```
$align-char-default as xs:string := ''
```

```
$align-char-pad := ' '
```

```
$align-char-width := 2
```

## Description

CALS tables support “char” alignment on columns. The most common use case for character alignment is to align a column of numbers on a decimal point even when the number of characters to right or left of the decimal point varies.

Three parameters apply to character alignment:

`$align-char-default`

Specifies the default alignment character.

`$align-char-pad`

Specifies the default padding character, usually an en space.

`$align-char-width`

Specifies the number of characters that follow the alignment character.

## Notes



1. The HTML table model doesn't really support character alignment. This feature pads each entry on the right as necessary to put the alignment character in the same place. Naturally, this will only produce the desired result if a monospace font is used in the cell.
2. If the alignment character appears more than once in the cell, the cells are aligned around the last occurrence. The notion of "last" and padding on the right or the left should be sensitive to the writing direction, but it isn't currently.
3. Character alignment is ignored if a cell contains markup.

*Example 1, “An example of char alignment” shows an example of character alignment on the “,” character.*

```

<informaltable frame="all">
  <tgroup cols="2" colsep="1" rowsep="1">
    <colspec column="2" align="char" char=","/>
    <tbody>
      <row>
        <entry>Row A</entry>
        <entry>10000,01</entry>
      </row>
      <row>
        <entry>Row B</entry>
        <entry>10</entry>
      </row>
      <row>
        <entry>Row C</entry>
        <entry>-3,14</entry>
      </row>
    </tbody>
  </tgroup>
</informaltable>

```

### Example 1. An example of char alignment

Row A	10000,01
Row B	10
Row C	-3,14

### Example 2. Tabular rendering

The alignment character can be specified on a per-column, per-table basis with the `char` attribute on `colspec`.

The alignment width and padding character can be specified on a per-column, per-table basis with a `db` processing instruction.

If the settings apply to the whole table, the processing instruction can be a child of `tgroup`, preceding any other elements. If you want to specify different values for different columns, the processing instruction must immediately follow the `colspec` for the column.

The `align-char-width` pseudo-attribute controls the number of characters following the alignment character. The `align-char-pad` pseudo-attribute controls the character used for padding.

A value that does not contain the alignment character is assumed to be followed immediately by the alignment character. (In other words, if you're aligning on “.”, “10” is considered to be “10.”.) If the pad character is a space<sup>1</sup>, then the pad character will be used to pad the value. If the pad character isn't a space, the value will be padded with the alignment character followed by pad characters as necessary.

---

<sup>1</sup> Technically, is in the class of Unicode characters considered to be spaces, one that matches `\p{Zs}`.

# \$allow-eval

Process the `eval` processing instruction?

Parameter:	{}allow-eval
Defined in:	param.xsl
Used in:	param.xsl, modules/inlines.xsl

## Synopsis

```
$allow-eval as xs:string := 'true'
```

## Description

If this parameter is true, XPath expressions contained in the `eval` processing instruction will be evaluated, otherwise the processing instruction is silently ignored.

If the eval is performed, the result of the evaluation is inserted into the formatted document.

# \$annotate-toc

Annotate the table-of-contents?

Parameter:	{}annotate-toc
Defined in:	param.xsl
Used in:	param.xsl, modules/toc.xsl

## Synopsis

```
$annotate-toc := 'true'
```

## Description

If true, a table of contents containing `refentry` elements will also include the `refpurpose` for each entry.

# \$annotation-collection

An external collection of annotations.

Parameter:	{annotation-collection}
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$annotation-collection as xs:string := "
```

## Description

Because annotations can be applied by pointing from the annotation to the element they annotate, they can be stored and managed externally. If `$annotation-collection` points to a collection of annotations, the stylesheets will add them to the document before it is processed.

To achieve this:

1. Create a collection of shared annotations and store them somewhere. The stylesheets will copy only the top-level `annotation` elements (`/* annotation`). The document element doesn't matter.
2. Use the `annotates` attribute on the annotations to point at elements in the document you're transforming.

Annotations can point in either direction, but you will get ID/IDREF validation errors if you attempt to point from the document into the annotation collection because the annotation collection won't be present when you validate. If your use case requires pointing in that direction, you will have better luck with a pipeline that combines the two documents before validating.

3. Run your transformation with `$annotation-collection` set to the URI of the document that contains your collection of annotations.

The stylesheets will apply annotations to the elements identified. Extra annotations, annotations that don't point to elements that actually exist in the document being transformed, will be ignored.

# \$annotation-mark

Identifying mark for annotations.

Parameter:	{annotation-mark}
Defined in:	param.xsl
Used in:	param.xsl, modules/annotations.xsl

## Synopsis

```
<xsl:param name="annotation-mark">  
  <sup>⊕</sup>  
</xsl:param>
```

## Description

When `annotation`s are rendered, the `$annotation-mark` is inserted at each location where an annotation occurs.

If JavaScript is used for annotations (see `$annotation-style`), clicking on the mark will bring up the annotation; if JavaScript is not used, the marks are numbered and the annotations appear as a form of footnote.

# \$annotation-placement

Determines where the annotation mark is placed.

Parameter:	{}annotation-placement
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$annotation-placement := 'after'
```

## Description

When `annotation`s are rendered, `$annotation-placement` determines where the `$annotation-mark` is placed. If the value is `after`, the mark or marks are placed at the very end of the element being annotated. If the value is `before`, the marks are placed at the very beginning of the element being annotated.

Each individual annotation can determine its placement by placing `before` or `after` in its `role` attribute.

# \$annotation-style

Selects the annotation style.

Parameter:	{annotation-style}
Defined in:	param.xsl
Used in:	param.xsl, modules/chunk-cleanup.xsl

## Synopsis

```
$annotation-style := 'javascript'
```

## Description

An `annotation` can be used to add annotations to arbitrary elements. Annotations are formatted something like footnotes, they appear at the bottom of the page and are linked from the point of the annotation. There are two annotation styles, “`javascript`” and “`inline`”.

If the annotation style is “`javascript`”, a script reference will be added. The script hides the annotations at the bottom of the page and instead renders them as modal dialogs when the marks are clicked. The presentation is accessible in the absence of JavaScript.

If the annotation style is “`inline`”, then the script is not included and the default presentation is used. No other values are supported at this time.

# \$annotations-js

Script to support popup (JavaScript) annotations.

Parameter:	{}annotations-js
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$annotations-js := 'js/annotations.js'
```

## Description

If popup (JavaScript) annotations appear in the document, a link to this script will be added to the document. This script must contain the JavaScript necessary to support the popup annotations feature.

# \$auto-toc

Automatically generate tables of contents.

Parameter:	{}auto-toc
Defined in:	param.xsl
Used in:	param.xsl, modules/toc.xsl
Since:	2.0.10

## Synopsis

```
$auto-toc as xs:string := 'true'
```

## Description

If this parameter is true, then a table of contents, or more generally, “lists of titles”, will be generated. If it is not true, such lists will only appear where an explicit `toc` element appears. An empty `toc` can be used to identify the place where an automatically generated list should appear. In either case, if the `toc` is not empty, then the hand-authored list appears instead of an automatically generated one.

Which elements will have an automatically generated list of titles is determined by how elements are processed in the `m:toc` mode.

# \$bibliography-collection

An external collection of bibliography entries.

Parameter:	{}bibliography-collection
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$bibliography-collection as xs:string := "
```

## Description

It's often convenient to share a common bibliography across many documents. If `$bibliography-collection` points to a bibliography, the stylesheets will automatically populate bibliography entries in the document being transformed.

To achieve this:

1. Create a shared bibliography and store it somewhere. Make sure that each entry in your shared bibliography has a unique `xml:id` value.
2. Run your transformation with `$bibliography-collection` set to the URI of that document.
3. In the document you're transforming, refer to the shared entries with completely empty `biblioentry` or `bibliomixed` elements that have the same `xml:id` as an entry in the shared bibliography.

The stylesheets will copy those entries into your document before processing it.

# \$bibliography-style

The bibliography style.

Parameter: {}bibliography-style

Defined in: param.xsl

Used in: param.xsl, modules/bibliography.xsl, modules/attributes.xsl

## Synopsis

\$bibliography-style as xs:string := 'default'

## Description

Bibliographies are complicated and many different styles exist. Creating properly formatted bibliographies from raw (<https://tdg.docbook.org/tdg/5.2/ch02.html#s.bibliography>) bibliography entries is especially challenging. (This parameter does not apply to formatting `bibliomixed` entries.)

default

The default style. This is controlled by the `biblioentry` template. See `$v:templates`. All `biblioentry` elements will be processed in `m:biblioentry` mode.

iso690

Attempts to implement ISO 690 bibliographic formatting. All `biblioentry` elements will be processed in `m:biblio690` mode. This is adapted from the examples in Bibliography according to ISO 690 and ISO 690-2 standards (<https://documents.docbook.org/iso690.html>).

The default style specified by `$bibliography-style` can be overridden on a case-by-case basis with the `db` processing instruction using the `bibliography-style` pseudo-attribute. This must appear in the `bibliography` or `bibliolist` that contains the entries.

# \$books-inherit-from

Identifies what inherited numeration applies to books.

Parameter:	{}books-inherit-from
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl

## Synopsis

```
$books-inherit-from as xs:string := "
```

## Description

If book numbers inherit from their ancestors, these are the ancestors they inherit from.

See *Section 3.5, “Controlling numeration”*.

# \$books-number-from

Identifies where book numeration begins.

Parameter:	{}books-number-from
Defined in:	param.xsl
Used in:	param.xsl, modules/numbers.xsl

## Synopsis

```
$books-number-from as xs:string := 'set'
```

## Description

Book numbers (if books are numbered) begin from here.

See *Section 3.5, “Controlling numeration”*.

root

Books are numbered from the beginning of the document.

set

Books are numbered from their parent set.

# \$callout-default-column

Default column for callouts.

Parameter:	{}callout-default-column
Defined in:	param.xsl
Used in:	param.xsl, modules/verbatim.xsl

## Synopsis

```
$callout-default-column := 60
```

## Description

When callouts are used in program listings (or screens), if the column is not specified for a callout, it will appear in this column.

# \$callout-unicode-start

Initial callout character.

Parameter: {}callout-unicode-start

Defined in: param.xsl

Used in: param.xsl, modules/programming.xsl, modules/verbatim.xsl

## Synopsis

```
$callout-unicode-start := 9311
```

## Description

When callouts are rendered, for example in `programlistingco`, this is the beginning of the range of characters to be used for callout “bugs”. This is the decimal value of the Unicode code point that is the “zero” point in the list, the one just before the first character.

The default value is 9311 which means the first callout will be 9312, U2460, “①”.

# \$chunk

Produce chunked output?

Parameter `{}`chunk

Defined in `param.xsl`

Used in: `main.xsl`, `param.xsl`, `modules/variable.xsl`, `modules/chunk-cleanup.xsl`

Used by: `$v:chunk`

## Synopsis

```
$chunk as xs:string? := ()
```

## Description

Specifying any value for this parameter turns on “chunking”, see *Section 2.6, ““Chunked” output”*. The value of the parameter is taken as the filename for the root chunk, unless some other mechanism identifies an alternate value.

# \$chunk-exclude

XPath expressions for excluding chunks.

Parameter:	{}chunk-exclude
Defined in:	param.xsl
Used in:	param.xsl, modules/chunk.xsl

## Synopsis

```
$chunk-exclude as xs:string* := ('self::db:partintro',
    'self::*[ancestor::db:partintro]',
    'self::db:annotation',
    'self::db:section[not(preceding-sibling::db:section)]',
    'self::db:sect1[not(preceding-sibling::db:sect1)]',
    'self::db:toc')
```

## Description

This parameter is only relevant when chunking is being performed, see [Section 2.6, ““Chunked” output”](#). This parameter contains a list of XPath expressions. When chunking is being applied, for any element that could be a chunk (see `$chunk-include`), each expression is evaluated with that element as the context item. If the effective boolean value of any expression is true, the element will not become a chunk.

The default value for this parameter is:

```
('self::db:partintro',
'self::*[ancestor::db:partintro]',
'self::db:toc')
```

In other words `partintro`, all of the descendants of `partintro`, and `toc` are explicitly excluded from being chunks.

The namespaces in `$chunk-filter-namespaces` will be in-scope when this expression is evaluated.

# \$chunk-include

XPath expressions for identifying potential chunks.

Parameter:	{}chunk-include
Defined in:	param.xsl
Used in:	param.xsl, modules/chunk.xsl

## Synopsis

```
$chunk-include as xs:string* := ('parent::db:set',
    'parent::db:book',
    'parent::db:part',
    'parent::db:reference',
    'self::db:refentry',
    'self::db:section',
    'self::db:sect1')
```

## Description

This parameter is only relevant when chunking is being performed, see [Section 2.6, ““Chunked” output”](#). This parameter contains a list of XPath expressions. When chunking is being applied, for every element in the document, each expression is evaluated with that element as the context item. If the effective boolean value of any expression is true, the element is a candidate for chunking. It will become a chunk unless `$chunk-exclude` excludes it or, in the special case of recursive sections, if it is nested too deeply.

The default value for this parameter is:

```
('parent::db:set',
'parent::db:book',
'parent::db:part',
'parent::db:reference',
'self::db:refentry',
'self::db:section',
'self::db:sect1')
```

In other words all of the direct children of `set`, `book`, `part`, `reference` will become chunks (the special case of `info` is automatically excluded); `refentry`, `section`, and `sect1` become chunks anywhere they appear. In the case of recursive sections, `$chunk-section-depth` also applies.

The namespaces in `$chunk-filter-namespaces` will be in-scope when this expression is evaluated.

# \$chunk-nav

Add keyboard navigation to chunks?

Parameter:	{}chunk-nav
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$chunk-nav as xs:string := 'true'
```

## Description

If `$chunk-nav` is true, keyboard navigation will be added to chunked output. This is accomplished by adding the `$chunk-nav-js` script to the generated HTML.

**Table 1. Default keyboard navigation keys**

Key	Navigation
<code>N</code> or <code>→</code>	Next sibling
<code>P</code> or <code>←</code>	Previous sibling
<code>U</code>	Parent
<code>H</code> or <code>Home</code>	Home
<code>S</code>	Toggle between normal and speaker view

Notes:

1. The `↑` used to be a shortcut for “parent”, like `U`, but that breaks the ability to navigate up and down the page with the arrow keys. It was removed in version 2.0.3.
2. The `S` functionality is really only relevant to when speaker notes are enabled.

# \$chunk-nav-js

Script to support keyboard navigation.

Parameter:	{}chunk-nav-js
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$chunk-nav-js as xs:string := 'js/chunk-nav.js'
```

## Description

If chunk navigation is enabled, this script is added to each chunk to support keyboard navigation.

# \$chunk-output-base-uri

Output base URI for chunks.

Parameter:	{}chunk-output-base-uri
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl

## Synopsis

```
<xsl:param  
    name="chunk-output-base-uri"  
    as="xs:string">  
  
<xsl:choose>  
    <xsl:when test="not($v:chunk)"><!-- it doesn't actually matter -->  
        <xsl:sequence select="" />  
    </xsl:when>  
    <xsl:when use-when="function-available('ext:cwd')" test="true()">  
        <xsl:sequence select="ext:cwd()"/>  
    </xsl:when>  
    <xsl:otherwise>  
        <xsl:message terminate="yes" select="You must specify the $chunk-output-base-uri"/>  
    </xsl:otherwise>  
</xsl:choose>  
</xsl:param>
```

## Description

This parameter identifies the output directory where “chunks” will be stored. If the stylesheet is not producing chunked results (see *Section 2.6, “Chunked output”*), this parameter has no effect.

If the `ext:cwd()` extension function (see *Section 2.5, “Extension functions”*) is available, its value will be the default value for this parameter. If the extension isn’t available, you must provide a value for this parameter.

## Important



If you're using the stylesheets from Maven, the `static-base-uri()` will be something like `https://cdn.docbook.org/release/...` and resolving the current working directory against that won't be useful. It's better in those cases to specify the parameter explicitly with a `file:` URI. (And note that it may need to be a `file:/path` URI, not a `file:///path` URI, in order to match correctly.)

# \$chunk-renumber-footnotes

Renumber footnotes when chunking?

Parameter:	{}chunk-renumber-footnotes
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:chunk-renumber-footnotes

## Synopsis

```
$chunk-renumber-footnotes := 'true'
```

## Description

If this parameter is true, footnotes will be renumbered within chunks. See  
\$v:chunk-renumber-footnotes .

# \$chunk-section-depth

Returns the chunking depth for recursive sections.

Parameter:	{}chunk-section-depth
Defined in:	param.xsl
Used in:	param.xsl, modules/chunk.xsl

## Synopsis

```
$chunk-section-depth := 1
```

## Description

This parameter is only relevant when chunking is being performed, see [Section 2.6, ““Chunked” output”](#). For recursive sections (`section`, and `refsection`), if the section would be a chunk according to `$chunk-include`, it is further tested against `$chunk-section-depth`. If it is nested more deeply than `$chunk-section-depth`, it will not be considered for chunking.

### Note



In principle, this parameter is unnecessary as it could be represented by `$chunk-exclude` expressions. However, changing the section depth is a common customization, so it's handled as a separate parameter.

# \$classsynopsis-indent

Indent for `classsynopsis` members.

Parameter:     `{}classsynopsis-indent`

Defined in:    `param.xsl`

Used in:       `param.xsl, modules/programming.xsl`

## Synopsis

```
$classsynopsis-indent := ' '
```

## Description

Elements inside a class synopsis (field and method declarations, for example) will be indented by this amount.

# \$component-numbers

Are components numbered?

Parameter:	{}component-numbers
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl
Used by:	\$v:title-groups

## Synopsis

```
$component-numbers as xs:string := '1'
```

## Description

If `$component-numbers` is true, components (chapters, appendixes, etc.) that do not have an explicit label will be labeled with their component number.

See *Section 3.5, “Controlling numeration”*.

# \$component-numbers-inherit

Include component labels in section labels?

## Synopsis

```
$component-numbers-inherit as xs:string := 'false'
```

## Description

This parameter is no longer used. See *Section 3.5, “Controlling numeration”*.

# \$components-inherit-from

Identifies what inherited numeration applies to components.

Parameter:	{}components-inherit-from
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl

## Synopsis

```
$components-inherit-from as xs:string := "
```

## Description

If component numbers inherit from their ancestors, these are the ancestors they inherit from.

See *Section 3.5, “Controlling numeration”*.

# \$components-number-from

Identifies where component numeration begins.

Parameter:	{}components-number-from
Defined in:	param.xsl
Used in:	param.xsl, modules/numbers.xsl

## Synopsis

```
$components-number-from as xs:string := 'book'
```

## Description

Component numbers (if components are numbered) begin from here.

See *Section 3.5, “Controlling numeration”*.

root

Components are numbered from the beginning of the document.

set

Components are numbered from their nearest ancestor set.

book

Components are numbered from their nearest ancestor book.

division

Components are numbered from their parent division.

# \$control-js

Script to support theme selection.

Parameter:	{}control-js
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$control-js as xs:string := 'js/controls.js'
```

## Description

If the `$theme-picker` is enabled, a link to this script will be added to the document. This script must contain the JavaScript necessary to support the theme controls.

# \$copyright-collapse-years

Collapse sequential copyright years into a range?

Parameter:	{copyright-collapse-years}
Defined in:	param.xsl
Used in:	param.xsl, modules/info.xsl

## Synopsis

```
$copyright-collapse-years := true()
```

## Description

If true, sequential copyright years will be collapsed into a range.

For example,

```
<copyright>
<year>1993</year>
<year>1997</year>
<year>1998</year>
<year>1999</year>
<holder>Jane Smith</holder>
</copyright>
```

might be rendered: “Copyright © 1993, 1997–1999 Jane Smith”.

If `$copyright-collapse-years` is false, that would be rendered: “Copyright © 1993, 1997, 1998, 1999 Jane Smith”.

See also `$copyright-year-range-separator` and `$copyright-year-separator`.

# \$copyright-year-range-separator

Separator character for copyright year ranges.

Parameter:	{copyright-year-range-separator}
Defined in:	param.xsl
Used in:	param.xsl, modules/info.xsl

## Synopsis

```
$copyright-year-range-separator := '-'
```

## Description

When a sequence of copyright years is collapsed into a range, this character is used to separate the first and last years.

See also [\\$copyright-collapse-years](#) and [\\$copyright-year-separator](#).

# \$copyright-year-separator

Separator character for copyright years.

Parameter:	{copyright-year-separator}
Defined in:	param.xsl
Used in:	param.xsl, modules/info.xsl

## Synopsis

```
$copyright-year-separator := ','
```

## Description

When a series of copyright years (or year ranges) is formatted, this string is used to separate them.

See also [\\$copyright-year-range-separator](#) and [\\$copyright-collapse-years](#).

# \$date-date-format, \$date-dateTime-format

Format strings for dates and dateTimes.

Parameter	{date-date-format
	{date-dateTime-format
Defined in	param.xsl (2)
Used in:	param.xsl, modules/functions.xsl, modules/info.xsl, modules/inlines.xsl
Used by:	f:date-format()

## Synopsis

```
$date-date-format := '[D01] [MNn,*-3] [Y0001]'
```

```
$date-dateTime-format := '[H01]:[m01] [D01] [MNn,*-3] [Y0001]'
```

## Description

The most convenient format for storing dates that will be machine processed: sorted, indexed, etc, is *ISO 8601*. When publishing these dates, it's often desirable to use a different format, one more familiar to readers.

A `pubdate` that conforms to an ISO 8601 date (“yyyy-mm-dd”) will be formatted with the `$date-date-format`; one that conforms to an ISO 8601 date/Time (“yyyy-mm-ddThh:mm:ss”) will be formatted with the `$date-dateTime-format`.

See also `f:date-format()`.

# \$dc-metadata

Output Dublin Core metadata?

Parameter:     {}dc-metadata

Defined in:    param.xsl

Used in:       param.xsl, modules/head.xsl

## Synopsis

```
$dc-metadata as xs:string := 'true'
```

## Description

If this parameter is true, the `head` element of each result document will contain `meta` elements for Dublin Core metadata that can be derived from the source document.

# \$debug

Debugging flags.

Parameter:	{}debug
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:debug
Static:	Yes

## Synopsis

```
$debug as xs:string := "
```

## Description

The `$debug` parameter contains a space separated list of flags. Depending on which flags are enabled, various additional debugging messages will be emitted during processing.

The following flags are supported:

callouts

Display additional information about callout processing.

cals-align-char

Display additional information about the computation of character alignment processing in CALS tables.

chunk-cleanup

Display additional information about chunk cleanup processing.

chunks

Display additional information about chunk selection.

**db4to5**

Display additional information about the DocBook 4.x to 5.x upgrade process.

**dynamic-profile**

Display additional information about dynamic profiling.

**dynamic-profile-suppress**

Display additional information about what elements are suppressed by dynamic profiling.

**image-properties**

Display additional information about the results of extracting properties from images. This will display all of the properties available from each image.

**image-markup**

Display the markup generated for media objects.

**intra-chunk-links**

Display additional information about the resolution of intra-chunk references. to links.

**intra-chunk-refs**

Display additional information about intra-chunk references.

**mediaobject-uris**

Display additional information about how media object URIs are constructed.

**linkbase**

Display additional information about how XLink linkbases are resolved.

**localization**

Display additional information about localization data.

numeration

Display additional information about how elements are numbered (in titles, in cross references, etc.)

objects

Display additional information about how media objects are selected.

pipeline

Display additional information about the pipeline processing stages.

profile

Display additional information about profiling.

profile-suppress

Display additional information about what elements are suppressed by profiling.

properties

Display additional information about the properties associated with a verbatim environment.

render-verbatim

Display additional information about how verbatim environments are rendered.

tables

Display additional information about how tables are rendered.

template-matches

Display additional information about how title page templates are selected.

templates

Display additional information about title page templates.

verbatim

Display additional information about how verbatim elements are processed.

**xlink**

Display additional information about XLink resolution.

# \$default-float-style

Default float style.

Parameter:	{}default-float-style
Defined in:	param.xsl
Used in:	param.xsl, modules/attributes.xsl

## Synopsis

```
$default-float-style := 'left'
```

## Description

The `floatstyle` attribute on formal objects (`figure`, etc.) indicates that they should float. This is achieved by adding the float style as a class value in the HTML output. If the value of `floatstyle` is “`float`”, then the `$default-float-style` will be applied. In other words, the CSS classes will be “`float left`” (as long as `$default-float-style` is “`left`”). If any other value is given for `floatstyle`, then the CSS classes will be “`float value`”.

# \$default-language

Default language.

Parameter: {}default-language

Defined in: param.xsl

Used in: param.xsl, modules/gentext.xsl, modules/functions.xsl

Used by: f:languages(), f:in-scope-language(), f:l10n-language()

## Synopsis

```
$default-language := 'en'
```

## Description

If the stylesheets need to generate text (the names of labels such as “Chapter”, for example), they attempt to do so in the same language as the document. This is determined by finding the nearest in-scope `xml:lang` attribute from the context where the generated text is required.

If there is no in-scope `xml:lang` attribute, or if there is no localization available for the language specified, the `$default-language` is used instead.

# \$default-length-magnitude, \$default-length-unit

The magnitude and units of the default length.

Parameter:	{}default-length-magnitude
	{}default-length-unit
Defined in:	param.xsl (2)
Used in:	param.xsl, modules/units.xsl
Used by:	f:parse-length()

## Synopsis

```
$default-length-magnitude := 25.0
```

```
$default-length-unit := '%'
```

## Description

If the stylesheet encounters a length that it cannot parse, the default magnitude and unit are used for that length. See *Section 5.2, “Lengths and units”*.

# \$default-theme

Default theme.

Parameter:	{}default-theme
Defined in:	param.xsl
Used in:	param.xsl, modules/chunk-cleanup.xsl

## Synopsis

```
$default-theme as xs:string := "
```

## Description

If a `$default-theme` is specified, its value will be added to the class attribute on the `html` element(s) generated. The actual theme implementation is performed in CSS. See `$theme-list` for a list of themes. You can add your own themes with custom CSS (with `m:html-head-links` mode, for example).

# \$division-numbers

Are divisions numbered?

Parameter:	{}division-numbers
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl
Used by:	\$v:title-groups

## Synopsis

```
$division-numbers as xs:string := '1'
```

## Description

If `$division-numbers` is true, divisions (books, parts, references, etc.) that do not have an explicit label will be labeled with their division number.

See also `$v:title-properties`.

# \$division-numbers-inherit

Include division labels in component labels?

## Synopsis

```
$division-numbers-inherit as xs:string := 'false'
```

## Description

This parameter is no longer used. See *Section 3.5, “Controlling numeration”*.

# \$divisions-inherit-from

Identifies what inherited numeration applies to divisions.

Parameter:	{}divisions-inherit-from
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl

## Synopsis

```
$divisions-inherit-from as xs:string := "
```

## Description

If division numbers inherit from their ancestors, these are the ancestors they inherit from.

See *Section 3.5, “Controlling numeration”*.

# \$divisions-number-from

Identifies where division numeration begins.

Parameter:	{}divisions-number-from
Defined in:	param.xsl
Used in:	param.xsl, modules/numbers.xsl

## Synopsis

```
$divisions-number-from as xs:string := 'book'
```

## Description

Divisions numbers (if divisions are numbered) begin from here.

See *Section 3.5, “Controlling numeration”*.

root

Divisions are numbered from the beginning of the document.

set

Divisions are numbered from their nearest ancestor set.

book

Divisions are numbered from their parent book.

# \$docbook-transclusion

Enable DocBook transclusion processing.

Parameter:	{}docbook-transclusion
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$docbook-transclusion := 'false'
```

## Description

The `$docbook-transclusion` controls whether or not DocBook transclusion processing (see *Transclusion*) is performed. The default at present is false because I don't believe it's in widespread use and transclusion processing on large documents can be quite slow. If either of those conditions change, the default may become true in some future version of the stylesheets.

# \$dynamic-profile-error

How are errors in dynamic profiling handled?

Parameter:	{}dynamic-profile-error
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$dynamic-profile-error := 'ignore'
```

## Description

If an error occurs evaluating a dynamic profiling expression, what should happen? That depends on the setting of `$dynamic-profile-error`:

`ignore`

The error is ignored. This has no effect on whether or not the element is included.

`include`

The error is ignored and the expression is considered to have returned a value of “true”.

`exclude`

The error is ignored and the expression is considered to have returned a value of “false”.

`error`

Raises the `dbe:DYNAMIC-PROFILE-EVAL-ERROR` exception.

any other value

Raises the `dbe:INVALID-DYNAMIC-PROFILE-ERROR` exception.

# \$dynamic-profile-variables

Dynamic profiling variables.

Parameter:	{}dynamic-profile-variables
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$dynamic-profile-variables as map(xs:QName, item()*)? := ()
```

## Description

The dynamic profile variables are passed to each of the pre- and post-processing transformations in addition to the standard parameters. See *Section 5.7, “The pre- and post-processing pipeline”*.

# \$dynamic-profiles

Is dynamic profiling enabled?

Parameter:	{}dynamic-profiles
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$dynamic-profiles as xs:string := 'false'
```

## Description

If this parameter is true, dynamic profiling will be applied to the document. See *Section 2.7.3, “Dynamic profiling”*.

# \$experimental-pmuj

Insert reverse links.

Parameter:	{experimental-pmuj}
Defined in:	param.xsl
Used in:	param.xsl, modules/xlink.xsl

## Synopsis

```
$experimental-pmuj := 'false'
```

## Description

If `$experimental-pmuj` is true, then links will be inserted from link targets back to link sources. In other words, if clicking on “A” jumps you to “B”, the stylesheets will add a link at “B” that jumps you back back to “A”. (Pmuj is jump spelled backwards.)

### Caution



This feature is entirely experimental. It may change in arbitrary ways or it may be abandoned.

Support is limited at the moment to inlines and formal objects that have title pages. There's currently no where to insert markup for every possible element that could be a link target.

Pmuj's from title pages should probably be inserted in the heading, not above it, just because the styling would be nicer.

# \$fallback-js

Script to support audio/video fallback.

Parameter:     {}fallback-js

Defined in:    param.xsl

Used in:      main.xsl, param.xsl, modules/objects.xsl

## Synopsis

```
$fallback-js := 'js/fallback.js'
```

## Description

Apparently, there's no declarative way to offer an HTML fallback message when all of the audio or video sources in a media object are unusable. Instead, it has to be done with JavaScript.

This parameter identifies the JavaScript file to load for this support. It must define a global function named `docbook_object_fallback` that takes the source node as a parameter.

To disable fallback, set this parameter to the empty sequence or the empty string.

Place this script in the `head` of the document and *do not* defer loading it.

# \$footnote-numeration

How are footnotes numbered?

Parameter:`{footnote-numeration}`

Defined in:`param.xsl`

Used in: `param.xsl, modules/footnotes.xsl, modules/chunk-cleanup.xsl`

## Synopsis

```
$footnote-numeration := ('1')
```

## Description

There are different styles of footnote numeration. This parameter contains a list of the symbols that should be used. If there could be more footnotes than symbols in the list, the last symbol in the list must be a character suitable for formatting numbers.

For example, if the sequence in `$footnote-numeration` is “\*”, “†”, “a”, then the first footnote will be marked with “\*”, the second with “†”, the third with “a”, the forth with “b”, and so on.

When chunking, footnotes may be renumbered within each chunk (see `$chunk-renumber-footnotes`).

# \$formal-object-title-placement

Specify placement of title on formal elements.

Parameter}formal-object-title-placement

Defined in param.xsl

Used in: param.xsl, modules/variable.xsl

Used by: \$v:formal-object-title-placement, \$v:formalgroup-nested-object-title-placement

## Synopsis

```
$formal-object-title-placement := 'after table:before formalgroup:before'
```

## Description

Formal elements are elements with a title that are often displayed with a caption: `figure`, `table`, `example` and `equation`. The `$formal-object-title-placement` parameter determines if the title precedes or follows the element itself.

The parameter consists of a series of tokens, “`before`”, “`after`”, or “`element : position`”. The named forms (e.g., `figure:before`) take precedence with the unnamed form serving as a default for any that don’t have named forms.

# \$formal-objects-inherit-from

Identifies what inherited numeration applies to formal-objects.

Parameter:	{}formal-objects-inherit-from
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl

## Synopsis

```
$formal-objects-inherit-from as xs:string := 'component'
```

## Description

If formal object numbers inherit from their ancestors, these are the ancestors they inherit from.

See *Section 3.5, “Controlling numeration”*.

# \$formal-objects-number-from

Identifies where formal-object numeration begins.

Parameter:     {}formal-objects-number-from

Defined in:    param.xsl

Used in:       param.xsl, modules/numbers.xsl

## Synopsis

```
$formal-objects-number-from as xs:string := 'component'
```

## Description

Formal object numbers (if formal objects are numbered) begin from here.

See *Section 3.5, “Controlling numeration”*.

root

Formal objects are numbered from the beginning of the document.

set

Formal objects are numbered from their nearest ancestor set.

book

Formal objects are numbered from their nearest ancestor book.

division

Formal objects are numbered from their nearest ancestor division.

component

Formal objects are numbered from their nearest ancestor component.

section

Formal objects are numbered from their parent section.

# \$formalgroup-nested-object-title-placement

Specify placement of title on formal elements inside `formalgroup` .

Parameter: `{$formalgroup-nested-object-title-placement}`

Defined in: `param.xsl`

Used in: `param.xsl`

Since: `2.0.13`

## Synopsis

```
$formalgroup-nested-object-title-placement := 'after'
```

## Description

The `$formal-object-title-placement` parameter controls how formal object titles are displayed generally. The `$formalgroup-nested-object-title-placement` parameter specifies how formal object titles are displayed when the formal object is inside a `formalgroup` . It has the same format as `$formal-object-title-placement` .

# \$funcsynopsis-default-style

Default style for function synopses.

Parameter:     {}funcsynopsis-default-style

Defined in:    param.xsl

Used in:       param.xsl, modules/programming.xsl

## Synopsis

```
$funcsynopsis-default-style := 'kr'
```

## Description

The `funcsynopsis` element can be rendered in several styles. The `$funcsynopsis-default-style` parameter determines which style is used by default.

# \$funcsynopsis-table-threshold

Maximum width of a function synopsis.

Parameter:     {}funcsynopsis-table-threshold

Defined in:    param.xsl

Used in:       param.xsl, modules/programming.xsl

## Synopsis

```
$funcsynopsis-table-threshold := 40
```

## Description

When rendering `funcsynopsis` elements, the width of the synopsis depends on the number of parameters and the lengths of their names and types. If the width of the `funcsynopsis` would exceed `$funcsynopsis-table-threshold` characters, the presentation switches from an inline style to a tabular style. The tabular style aligns the parameters up vertically after the function name.

# \$funcsynopsis-trailing-punctuation

Trailing punctuation in function synopses.

Parameter:     {}funcsynopsis-trailing-punctuation

Defined in:    param.xsl

Used in:       param.xsl, modules/programming.xsl

## Synopsis

```
$funcsynopsis-trailing-punctuation := ';'
```

## Description

This punctuation character is added after the end of a `funcsynopsis`. The default value of “;” is appropriate for C and other languages that use “;” as statement separator.

# \$generate-html-page

Generate the HTML page structure around the styled document.

Parameter:	{}generate-html-page
Defined in:	param.xsl
Used in:	docbook.xsl, param.xsl
Used by:	t:docbook

## Synopsis

```
$generate-html-page as xs:string := 'true'
```

## Description

If this parameter is true, then a complete HTML page will be generated for the transformed document: an `html` tag, `head` and `body` tags, etc. If it's false, then only the “raw” transformed content will be produced. This is true for both the primary output and any secondary result documents.

### Note



When the “raw” output option is selected, links to the CSS stylesheets, scripts, and other interactive features will not be generated. You must make sure those are provided in some other way.

# \$generate-index

Automatically generate an index?

Parameter:      {}generate-index

Defined in:      param.xsl

Used in:      param.xsl, modules/index.xsl

## Synopsis

```
$generate-index := 'true'
```

## Description

If a document contains an empty `setindex`, `index`, or `indexdiv`, and `$generate-index` is true, then an index will automatically be generated from any `indexterm`s found in the document.

# \$generate-nested-toc

Determines if subsubections appear in the table of contents.

## Synopsis

```
$generate-nested-toc as xs:string := 'not(f:section(.))  
or (f:section(.) and f:section-depth(.) le $vp:section-toc-depth)'
```

## Description

This parameter is no longer used. Instead, you can control the elements that appear in a table of contents with the `m:toc-nested` mode.

# \$generate-toc

Generate a table-of-contents?

## Synopsis

```
$generate-toc as xs:string := '(empty(parent::*) and self::db:article)
                                or self::db:set or self::db:book
                                or self::db:part or self::db:reference'
```

## Description

This parameter is no longer used. Instead, you can control the elements that contain a table of contents with the `m:toc` mode.

# \$generate-trivial-toc

Generate a trivial table-of-contents?

Parameter:	{}generate-trivial-toc
Defined in:	param.xsl
Used in:	param.xsl, modules/toc.xsl

## Synopsis

```
$generate-trivial-toc as xs:string := 'false'
```

## Description

If the `$generate-trivial-toc` parameter is true, a table of contents will be generated even if it consists of only a single entry. Otherwise, such “trivial” tables of contents will be elided.

This parameter applies only to top-level tables of contents. Nested tables of contents will be generated even if they consist of only a single entry.

# \$generated-id-root

The string generated as the unique identifier for the root of the tree.

Parameter:     {}generated-id-root

Defined in:    param.xsl

Used in:       param.xsl, modules/functions.xsl

Used by:       f:generate-id()

## Synopsis

```
$generated-id-root := 'R'
```

## Description

When `f:generate-id()` is constructing a unique identifier for an element, it walks up the ancestors of that element. If it reaches the root of the tree, it uses `$generated-id-root` to mark that location.

# \$generated-id-sep

The string generated to separate parts of a unique identifier.

Parameter:	{}generated-id-sep
Defined in:	param.xsl
Used in:	param.xsl, modules/functions.xsl
Used by:	f:generate-id()

## Synopsis

```
$generated-id-sep := '_'
```

## Description

When `f:generate-id()` is constructing a unique identifier for an element, it walks up the ancestors of that element. To make the identifiers easier to read, and to disambiguate values, it places `$generated-id-sep` between each component that it uses.

### Note



Historically, a “.” was used as the separator. However, if JavaScript is going to be used to process the HTML documents that the stylesheet produces, that may be inconvenient. (Because selectors use “.” to separate element names from class values.)

# \$generator-metadata

Output generator metadata?

Parameter:	{}generator-metadata
Defined in:	param.xsl
Used in:	param.xsl, modules/head.xsl

## Synopsis

```
$generator-metadata as xs:string := 'true'
```

## Description

If this parameter is true, the `head` element of each result document will contain a `meta` element with the name `generator` that identifies the stylesheet and processor used to produce the result.

# \$gentext-language

Specifies language for generated text.

Parameter: {}gentext-language

Defined in: param.xsl

Used in: param.xsl, modules/gentext.xsl, modules/functions.xsl

Used by: f:languages(), f:l10n-language()

## Synopsis

```
$gentext-language := ()
```

## Description

The language used for generated text (words “Chapter” and “Figure”, for example) usually depends on the language of the (section of) the document where they appear. If `$gentext-language` is specified, that language will be used for all generated text, regardless of the context.

# \$glossary-collection

An external collection of glossary entries.

Parameter:	{glossary-collection}
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$glossary-collection as xs:string := "
```

## Description

It's often convenient to share a common glossary across many documents. If `$glossary-collection` points to a glossary, the stylesheets will automatically populate glossary entries in the document being transformed.

To achieve this:

1. Create a shared glossary and store it somewhere. Make sure that each entry in your shared glossary has a unique `xml:id` value.
2. Run your transformation with `$glossary-collection` set to the URI of that document.
3. In the document you're transforming, refer to the shared entries with completely empty `glossentry` elements that have the same `xml:id` as an entry in the shared glossary.

The stylesheets will copy those entries into your document before processing it.

# \$glossary-sort-entries

Sort glossaries?

Parameter:	{}glossary-sort-entries
Defined in:	param.xsl
Used in:	param.xsl, modules/glossary.xsl

## Synopsis

```
$glossary-sort-entries := true()
```

## Description

If `$glossary-sort-entries` is true, the entries in a `glossary` or `glosslist` will be sorted before transformation. This saves the author from the burden of maintaining the list in a strictly alphabetic order.

# \$html-extension

The extension used for HTML output when chunking.

Parameter:     {}html-extension

Defined in:    param.xsl

Used in:       param.xsl, modules/chunk.xsl

Used by:       f:chunk-filename()

## Synopsis

```
$html-extension := '.html'
```

## Description

When using chunking (see *Section 2.6, ““Chunked” output”*), the resulting HTML documents will have this extension by default.

# \$image-ignore-scaling

Ignore scaling?

Parameter `{image-ignore-scaling}`

Defined in `param.xsl`

Used in: `param.xsl`, `modules/objects.xsl`

Used by: `f:object-width()`, `f:object-height()`, `f:object-contentwidth()`, `f:object-contentheight()`, `f:object-scalefit()`, `f:object-scale()`

## Synopsis

`$image-ignore-scaling` as `xs:boolean := false()`

## Description

If this parameter is true, all of the scaling attributes on images are ignored. Images will be displayed at their intrinsic size.

# \$image-nominal-height

Nominal height of an image.

Parameter:	{}image-nominal-height
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:image-nominal-height

## Synopsis

```
$image-nominal-height := '4in'
```

## Description

If the extension functions necessary to determine the intrinsic height of an image are unavailable, or if the height cannot be determined, this value will be used as the assumed intrinsic height of the image.

# \$image-nominal-width

Nominal width of an image.

Parameter:	{}image-nominal-width
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:image-nominal-width

## Synopsis

```
$image-nominal-width := $nominal-page-width
```

## Description

If the extension functions necessary to determine the intrinsic width of an image are unavailable, or if the width cannot be determined, this value will be used as the assumed intrinsic width of the image.

# \$image-property-warning

Warn if image properties cannot be obtained?

Parameter:	{}image-property-warning
Defined in:	param.xsl
Used in:	param.xsl, modules/objects.xsl
Used by:	f:object-properties()

## Synopsis

```
$image-property-warning := true()
```

## Description

This flag controls whether or not an `xsl:message` is emitted when the `ext:image-properties()` extension function is unavailable.

# \$index-on-role, \$index-on-type

Make distinct indexes based on role and type?

Parameter:	{}index-on-type
	{}index-on-role
Defined in:	param.xsl (2)
Used in:	param.xsl, modules/index.xsl
Used by:	t:generate-index

## Synopsis

```
$index-on-type := 'true'
```

```
$index-on-role := 'true'
```

## Description

DocBook supports the creation of different types of index. A book that documents an API might have a general index, for example, and also an index of functions, an index of variables, etc.

Typed indexes are created by specifying the index type in the `type` or `role` attribute. If `$index-on-type` is true, then separate indexes will be generated based on `type` values. If `$index-on-role` is true, then separate indexes will be generated based on `role` values.

# \$index-show-entries

Make index entries visible in the text?

Parameter:	{}index-show-entries
Defined in:	param.xsl
Used in:	param.xsl, modules/index.xsl

## Synopsis

```
$index-show-entries := ()
```

## Description

If this value is non-empty, small markers will be left in the text where `indexterm` elements appear. This is probably not appropriate for final publication, but it can be a useful way to review the level of indexing.

# \$indexed-section-groups

Generate index entries grouped by section.

Parameter:	{}indexed-section-groups
Defined in:	param.xsl
Used in:	param.xsl, modules/index.xsl

## Synopsis

```
$indexed-section-groups := 'true'
```

## Description

Before the introduction of this parameter, an automatically generated index contained only one index entry for any given section. If the `$indexed-section-groups` parameter is true, an index entry will be generated for every `indexterm`, with a `span` surrounding all of the entries for each section. This results in a more complete index while still preserving the ability to see in which sections the terms occur.

# \$lists-of-equations

Generate a list of equations?

Parameter:	{lists-of-equations}
Defined in:	param.xsl
Used in:	param.xsl, modules/toc.xsl

## Synopsis

```
$lists-of-equations as xs:string := 'false'
```

## Description

If true, a list of equations will be generated. By default, they are only generated for `book`s and `set`s.

This is a pseudo-boolean parameter. It is considered true if it has the value '1', 'yes', or 'true'. Any other value is considered false.

# \$lists-of-examples

Generate a list of examples?

Parameter:	{lists-of-examples}
Defined in:	param.xsl
Used in:	param.xsl, modules/toc.xsl

## Synopsis

```
$lists-of-examples as xs:string := 'true'
```

## Description

If true, a list of examples will be generated. By default, they are only generated for `book`s and `set`s.

This is a pseudo-boolean parameter. It is considered true if it has the value '1', 'yes', or 'true'. Any other value is considered false.

# \$lists-of-figures

Generate a list of figures?

Parameter:     {}lists-of-figures

Defined in:    param.xsl

Used in:       param.xsl, modules/toc.xsl

## Synopsis

```
$lists-of-figures as xs:string := 'true'
```

## Description

If true, a list of figures will be generated. By default, they are only generated for `book`s and `set`s.

This is a pseudo-boolean parameter. It is considered true if it has the value '1', 'yes', or 'true'. Any other value is considered false.

# \$lists-of-procedures

Generate a list of figures?

Parameter:	{}lists-of-procedures
Defined in:	param.xsl
Used in:	param.xsl, modules/toc.xsl

## Synopsis

```
$lists-of-procedures as xs:string := 'false'
```

## Description

If true, a list of procedures will be generated. By default, they are only generated for `book`s and `set`s.

This is a pseudo-boolean parameter. It is considered true if it has the value '1', 'yes', or 'true'. Any other value is considered false.

# \$lists-of-tables

Generate a list of tables?

Parameter:     {}lists-of-tables

Defined in:    param.xsl

Used in:       param.xsl, modules/toc.xsl

## Synopsis

```
$lists-of-tables as xs:string := 'true'
```

## Description

If true, a list of tables will be generated. By default, they are only generated for `book`s and `set`s.

This is a pseudo-boolean parameter. It is considered true if it has the value '1', 'yes', or 'true'. Any other value is considered false.

# \$local-conventions

Transformation for local conventions.

Parameter:	{}local-conventions
Defined in:	param.xsl
Used in:	docbook.xsl, param.xsl
Used by:	\$v:standard-transforms

## Synopsis

```
$local-conventions as xs:string? := ()
```

## Description

This parameter allows you to specify a transformation for local markup conventions. This book uses several non-DocBook tagging conventions as a typing convenience, `<att>` for `<tag class="attribute">`, for example. These can be translated back into proper DocBook markup by the `$local-conventions` stylesheet. This stylesheet is run during the `$v:standard-transforms`, just before validation (see `$relax-grammar`).

For example, the test suite stylesheet that transforms pseudo-DocBook elements “`att`” and “`mode`” into valid DocBook markup looks like this:

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://docbook.org/ns/docbook"
    xmlns:db="http://docbook.org/ns/docbook"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="db xs"
    version="3.0">

<xsl:template match="db:att">
    <tag class="attribute">
        <xsl:copy-of select="@* except @class"/>
        <xsl:apply-templates/>
    </tag>
</xsl:template>

<xsl:template match="db:mode">
    <code role="{string-join(
        distinct-values(@role/string(), 'mode'),
        ' ')}">
        <xsl:copy-of select="@* except @role"/>
        <xsl:apply-templates/>
    </code>
</xsl:template>

<xsl:template match="element()">
    <xsl:copy>
        <xsl:apply-templates select="@*,node()"/>
    </xsl:copy>
</xsl:template>

<xsl:template match="attribute()|text()|comment()
    |processing-instruction()">
    <xsl:copy/>
</xsl:template>
</xsl:stylesheet>
```

# \$mathml-js

JavaScript library supporting MathML.

Parameter:                    {}mathml-js

Defined in:                    param.xsl

Used in:                    main.xsl, param.xsl

## Synopsis

```
$mathml-js := 'https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.1/MathJax.js?config=MML_CHTM'
```

## Description

If your documents use MathML, this library will be included to support MathML rendering. *MathJax* is a popular choice.

# \$mediaobject-accessibility

Accessibility settings for media objects.

Parameter:	{}mediaobject-accessibility
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl

## Synopsis

```
$mediaobject-accessibility as xs:string := 'summary details'
```

## Description

The `$mediaobject-accessibility` parameter determines how accessibility features are added to media objects (image, video, and audio elements).

The value of this parameter is a space-separated list of strings. If the list contains:

summary

A `summary` attribute will be added if there is an `alt` element or a `textobject` containing a single `phrase` is available.

details

A `details` element will be added to the `div` that surrounds the image if there is a `textobject` containing anything other than a single `phrase` available.

a11y-metadata

Metadata will be provided using the conventions described in EPUB Accessibility (<https://www.w3.org/TR/epub-a11y/>).

The list may contain either or both values. If other values are present, they are ignored.

See *Example 1, “An example of media object accessibility”*.

```

<mediaobject>
  <alt>This is a skeuomorphic “postage stamp” with the
  DocBook logo.</alt>
  <imageobject>
    <imagedata fileref="media/stamp400x256.png"/>
  </imageobject>
  <textobject>
    <para>This is a skeuomorphic image of a “postage
    stamp.” Centrally, it features the DocBook wood
    duck logo. the word “DocBook” appears on the left
    hand side, rotated 90° counter-clockwise. The
    DocBook tag line, “The Source for Documentation”
    is printed in the upper-right corner.
  </para>
  </textobject>
</mediaobject>

```

### Example 1. An example of media object accessibility

One possible rendering of such an example:



Inspection of the HTML will reveal that the `summary` attribute is present on the `div` that wraps the figure and a `details` element precedes the image. These

may or may not be rendered by your user agent depending on its accessibility features and settings.

See also [\\$table-accessibility](#).

# \$mediaobject-details-placement

Specify placement of details on media objects.

Parameter:	{}mediaobject-details-placement
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:mediaobject-details-placement
Since:	2.0.13

## Synopsis

```
$mediaobject-details-placement := 'before'
```

## Description

If a details element is generated for a media object, this parameter determines if it goes before or after the object.

The parameter consists of a series of tokens, “before”, “after”, or “*element : position*”. The named forms (e.g., `figure:before`) take precedence with the unnamed form serving as a default for any that don’t have named forms.

# \$mediaobject-exclude-extensions

Exclude media objects by extension.

Parameter: `{}mediaobject-exclude-extensions`

Defined in: `param.xsl`

Used in: `param.xsl, modules/variable.xsl`

Used by: `$v:mediaobject-exclude-extensions`

## Synopsis

```
$mediaobject-exclude-extensions as xs:string := ".eps .ps .pdf"
```

## Description

The `mediaobject` element (and `inlinemediaobject`) can contain several, alternative objects. The `$mediaobject-exclude-extensions` parameter provides a way to exclude some of them.

One common use case is to provide two versions of each image, one as a bitmap, perhaps a PNG ([https://en.wikipedia.org/wiki/Portable\\_Network\\_Graphics](https://en.wikipedia.org/wiki/Portable_Network_Graphics)) , and another as a vector, perhaps an EPS ([https://en.wikipedia.org/wiki/Encapsulated\\_PostScript](https://en.wikipedia.org/wiki/Encapsulated_PostScript)) . Then `$mediaobject-exclude-extensions` can be used to exclude “.eps” images from the web format and “.png” images from the print (or paged media) format.

# \$mediaobject-grouped-by-type

Are media grouped by type?

Parameter:	{}mediaobject-grouped-by-type
Defined in:	param.xsl
Used in:	param.xsl, modules/objects.xsl
Used by:	f:mediaobject-amend-uri()
Since:	2.0.8

## Synopsis

```
$mediaobject-grouped-by-type as xs:string := 'false'
```

## Description

If `$mediaobject-grouped-by-type` is true, an extra directory level is implicit in the input path. Images are grouped by type, so `image.png` is assumed to be in `$mediaobject-input-base-uri/png/image.png`.

The `f:mediaobject-amend-uri()` function adds the type, which is computed by `f:mediaobject-type()`.

# \$mediaobject-input-base-uri

Base URI of images and other media in the XML sources.

Parameter:	{}mediaobject-input-base-uri
Defined in:	param.xsl
Used in:	param.xsl, modules/objects.xsl
Used by:	f:mediaobject-input-base-uri()

## Synopsis

```
$mediaobject-input-base-uri as xs:string? := ()
```

## Description

Computing the correct URI for media objects (images, videos, audio files, etc.) is complicated. See *Section 5.4.1, “Mediaobject URIs”*. The `$mediaobject-input-base-uri` is used to compute the absolute base URI of input media objects.

If the “cwd” extension function is available, then the default value of this parameter is the current working directory. Otherwise, the default value is the empty string.

The current working directory is only likely to be a useful value for this parameter if you are running the processor in the same directory where your XML source files are stored. See `$v:mediaobject-input-base-uri`.

# \$mediaobject-output-base-uri

Base URI of images and other media in the output.

Parameter: {}mediaobject-output-base-uri

Defined in: param.xsl

Used in: param.xsl, modules/variable.xsl, modules/chunk-cleanup.xsl

Used by: \$v:mediaobject-output-base-uri

## Synopsis

```
$mediaobject-output-base-uri as xs:string? := ()
```

## Description

Computing the correct URI for media objects (images, videos, audio files, etc.) is complicated. See *Section 5.4.1, “Mediaobject URIs”*. The `$mediaobject-output-base-uri` is used to compute the base URI of media objects in the output.

It defaults to the empty string. See `$v:mediaobject-output-base-uri`.

# \$mediaobject-output-paths

Preserve input mediaobject paths in output.

Parameter:     {}mediaobject-output-paths

Defined in:    param.xsl

Used in:      param.xsl, modules/chunk-cleanup.xsl

## Synopsis

```
$mediaobject-output-paths as xs:string := 'true'
```

## Description

FIXME:

# \$mediaobject-video-element

Use the video element?

Parameter:	{}mediaobject-video-element
Defined in:	param.xsl
Used in:	param.xsl, modules/objects.xsl

## Synopsis

```
$mediaobject-video-element as xs:string := 'video'
```

## Description

This element identifies the element to use for video content. The default starting in version 1.11.0 is `video` where it had previously been `iframe`.

Video is a bit complicated, the `video` element works best for local video sources but won't work for embedding content from other sites. If all of your content is embedded, specifying `iframe` here will be simplest.

If you need a mixture, of styles, it can be specified on a per-video basis with the `db` processing instruction using the `video` pseudo-attribute.

### Note



The only supported values are `iframe` and `video`; you cannot use this parameter or the processing instruction to insert arbitrary element names.

# \$message-level

How chatty should status messages be?

Parameter: {}message-level

Defined in: param.xsl

Used in: param.xsl, modules/verbatim.xsl, modules/chunk-cleanup.xsl

## Synopsis

\$message-level as xs:integer := 1

## Description

Broadly speaking, there are three kinds of messages that the stylesheets produce: debugging messages, status messages, and errors. Debugging messages are controlled by the `$debug` parameter. Error messages cause stylesheet processing to abort. Status messages fall somewhere in between. They alert you to facts about your document that you might want to fix, for example, broken cross references. If you ignore them, the stylesheets will produce some output.

Previously, status messages were always printed. This parameter has been introduced to provide more control. If `$message-level` is 0, most status messages will be suppressed. A value greater than 0 will cause some or all status messages to be printed. (At the time of this writing, all messages are at level 1, but it's possible that more detailed messaging will be introduced in the future.)

# \$nominal-page-width

The nominal page width.

Parameter:	{}nominal-page-width
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$image-nominal-width, \$v:nominal-page-width

## Synopsis

```
$nominal-page-width := '6in'
```

## Description

To calculate the width of the columns in some complex CALS tables, the stylesheets need to know the page width. The `$nominal-page-width` is used for this value.

# \$number-single-appendix

Number a single appendix?

Parameter:	{}number-single-appendix
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl

## Synopsis

```
$number-single-appendix := 'true'
```

## Description

If a book (or other element) has several appendixes, they will be numbered, usually “A”, “B”, “C”, etc. If there is only a single appendix, it will be numbered if `$number-single-appendix` is true. Otherwise, it will be unnumbered.

This can be used to create the numbered title “Appendix” for a single appendix.

# \$olink-databases

External olink databases.

Parameter:	{}olink-databases
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:olink-databases

## Synopsis

```
$olink-databases as xs:string := "
```

## Description

The `$olink-databases` parameter is a comma separated list of URIs. Each URI should contain an `olink` target database.

# \$orderedlist-item-numeration

Numeration for nested ordered lists.

Parameter:	{}orderedlist-item-numeration
Defined in:	param.xsl
Used in:	param.xsl, modules/functions.xsl
Used by:	f:orderedlist-item-numeration()

## Synopsis

```
$orderedlist-item-numeration := '1aiAl'
```

## Description

The `$orderedlist-item-numeration` parameter controls the enumeration style of nested `orderedlist` elements. Items in the top-level list will use the enumeration style of the first character in `$orderedlist-item-numeration`, items in the second-level list will use the enumeration style of the second character in the string, etc. If the list depth exceeds the number of characters in the `$orderedlist-item-numeration`, selection “wraps back around” to the first character.

In other words, if the string is “1ai”: list items at the first, fourth, seventh, tenth, etc. depth will have arabic enumeration. List items at the second, fifth, eighth, eleventh, etc. depth will have lowercase alpha enumeration, etc.

# \$othername-in-middle

Treat othername as a middle name.

Parameter:	{}othername-in-middle
Defined in:	param.xsl
Used in:	param.xsl, modules/info.xsl
Used by:	t:person-name-first-last

## Synopsis

```
$othername-in-middle := 'true'
```

## Description

If `$othername-in-middle` is true, then the first `othername` in a `personname` will be presented as the person's "middle" name. See `t:person-name`.

# \$output-media

The intended output media.

Parameter:	{}output-media
Defined in:	main.xsl, param.xsl
Used in:	param.xsl, modules/head.xsl

## Synopsis

```
$output-media := 'screen'
```

## Description

The `$output-media` parameter identifies the intended output medium, `screen` or `print`. It can be used to make conditional transformations.

# \$oxy-markup

Is Oxygen change tracking markup rendered?

Parameter:	{oxy-markup}
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$oxy-markup := 'false'
```

## Description

If `$oxy-markup` is true, then the change tracking processing instructions that Oxygen inserts into the document will be transformed into elements marked with CSS classes for rendering in the output.

Rendering Oxygen change markup can be specified on a document-by-document basis with a `db` processing instruction. If the `oxy-markup` pseudo-attribute is true, the markup will be rendered. This processing instruction must be in the `info` element of the document element.

# \$page-style

Select the page style.

Parameter:	{page-style}
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$page-style as xs:string := 'article'
```

## Description

This value selects the page style. The page style is reflected in the `class` attribute on the root `html` element. It is then used by the CSS stylesheets. A page style of “`value`” will be added to the class attribute as “`value-style`”.

The CSS print stylesheets support two styles: `article` and `book`. The `book` style places new components on a right-hand page and adjusts the inner margin for binding.

# \$pagetoc-dynamic

Make the on-page ToC dynamic?

Parameter:	{}pagetoc-dynamic
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$pagetoc-dynamic := 'true'
```

## Description

If this value is true, and the on-page ToC is displayed, then it will be updated dynamically to reflect the readers position. This has two consequences. First, the section titles that are currently in view will be highlighted. The default highlighting is to make them a little darker and place a bullet next to them. The highlighting can be changed with CSS. Second, a clickable symbol is added in the upper left corner. This lets the reader toggle between dynamic, non-dynamic, and hidden views.

If the on-page ToC is not dynamic, then no highlighting is performed.

The clickable symbols can be changed as well, but it requires adding a bit more JavaScript to the page. Ensure that your code runs first and set the following properties:

```
1 window.DocBook = {};
  window.DocBook.pagetoc = {};
  window.DocBook.pagetoc.decorated = "ON";
  window.DocBook.pagetoc.plain = "OFF";
5  window.DocBook.pagetoc.hidden = "HIDDEN";
```

The values can be strings or markup, such as an `img` element.

# \$pagetoc-elements

Elements that should have an on-page ToC.

Parameter:	{}pagetoc-elements
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl

## Synopsis

```
$pagetoc-elements := "
```

## Description

This parameter is a space-separated list of element names (local names). An on-page ToC will be generated for elements with these names.

# \$pagetoc-js

Script to support the on-page ToC.

Parameter:	{pagetoc-js}
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$pagetoc-js := 'js/pagetoc.js'
```

## Description

If the on-page ToC is enabled, a link to this script will be added to the document. This script must contain the JavaScript necessary to support the on-page ToC feature.

# \$paper-size

Select the paper style.

Parameter:	{}paper-size
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$paper-size as xs:string? := ()
```

## Description

This value selects the paper size. The paper size is reflected in the `class` attribute on the root `html` element. It is then used by the CSS stylesheets.

The CSS print stylesheets support three page sizes: `A4`, `A5`, and (US) `Letter`. Additional sizes can be implemented in CSS.

# \$persistent-toc

Generate a persistent Table of Contents?

Parameter: persistent-toc

Defined in: param.xsl

Used in: main.xsl, param.xsl, modules/head.xsl, modules/chunk-cleanup.xsl,  
modules/chunk-output.xsl

## Synopsis

```
$persistent-toc := 'false'
```

## Description

If this value is true, then a persistent ToC will be generated.

# \$persistent-toc-css

Persistent ToC CSS.

Parameter:	{}persistent-toc-css
Defined in:	param.xsl
Used in:	param.xsl, modules/head.xsl

## Synopsis

```
$persistent-toc-css := 'css/docbook-toc.css'
```

## Description

If the persistent ToC popup is enabled, a link to this CSS file will be added to the HTML HEAD.

# \$persistent-toc-filename

Name of file that holds the persistent ToC.

Parameter:	{}persistent-toc-filename
Defined in:	param.xsl
Used in:	param.xsl, modules/chunk-output.xsl
Since:	2.0.2

## Synopsis

```
$persistent-toc-filename as xs:string? := 'persistent-toc.html'
```

## Description

When the `$persistent-toc` parameter is true, the persistent ToC will be stored either in each HTML file, or in a single file that is loaded dynamically. If `$persistent-toc-filename` is not empty, then it will be used as the name for an external file that will be loaded when the ToC is requested.

# \$persistent-toc-js

Script to support the persistent ToC.

Parameter:	{}persistent-toc-js
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$persistent-toc-js := 'js/persistent-toc.js'
```

## Description

If the persistent ToC is enabled, a link to this script will be added to the document. This script must contain the JavaScript necessary to support the persistent ToC feature.

# \$persistent-toc-search

Generate a search box in the persistent ToC.

Parameter:	{ }persistent-toc-search
Defined in:	param.xsl
Used in:	param.xsl, modules/chunk-output.xsl

## Synopsis

```
$persistent-toc-search := 'true'
```

## Description

If this value is true, then a search box is added to the top of the persistent Table of Contents (ToC). Any string typed into this search box will be used to filter the entries shown below it.

The filter is a case-insensitive regular expression match against the text of each line in the ToC where arbitrary characters may occur anywhere in the string. In other words, the text “cat” will generate the regular expression “c.\*a.\*t.\*” for the purpose of searching.

This setting has no effect if `$persistent-toc` is false.

# \$personal-name-style

The style for formatting personal names.

Parameter:	{personal-name-style}
Defined in:	param.xsl
Used in:	param.xsl, modules/info.xsl

## Synopsis

```
$personal-name-style := ()
```

## Description

The stylesheets can format personal names in a variety of ways. This is usually locale dependent. If `$personal-name-style` is specified, it will be used in preference to the locale dependent style. See `t:person-name`.

# \$pixels-per-inch

The number of pixels per inch.

Parameter:	{}pixels-per-inch
Defined in:	param.xsl
Used in:	param.xsl, modules/units.xsl
Used by:	\$v:unit-scale

## Synopsis

```
$pixels-per-inch := 96.0
```

## Description

If the stylesheets need to convert between absolute and relative units (see *Section 5.2, “Lengths and units”*), this value is used to convert lengths into pixels. If `$pixels-per-inch` has the value 96, then 25% of a 6in width is 144px.

# \$procedure-step-numeration

Numeration for nested procedure steps.

Parameter:	{procedure-step-numeration}
Defined in:	param.xsl
Used in:	param.xsl, modules/functions.xsl
Used by:	f:step-numeration()

## Synopsis

```
$procedure-step-numeration := '1aiAl'
```

## Description

The `$procedure-step-numeration` parameter controls the enumeration style of nested steps and substeps in a `procedure`. Steps at the top level will use the enumeration style of the first character in `$procedure-step-numeration`, substeps at the second-level will use the enumeration style of the second character in the string, etc. If the step depth exceeds the number of characters in the `$procedure-step-numeration`, selection “wraps back around” to the first character.

In other words, if the string is “1ai”: steps at the first, fourth, seventh, tenth, etc. depth will have arabic numeration. Steps at the second, fifth, eighth, eleventh, etc. depth will have lowercase alpha numeration, etc.

# \$productionset-lhs-rhs-separator

Separator in productions.

Parameter:	{}productionset-lhs-rhs-separator
Defined in:	param.xsl
Used in:	param.xsl, modules/programming.xsl

## Synopsis

```
$productionset-lhs-rhs-separator := ':='
```

## Description

A `productionset` consists of non-terminals on the “left hand side” (LHS) and the productions that they expand to on the “right hand side” (RHS). When formatted, this string will be used as the separator between the left- and right-hand sides.

# \$profile-arch

Profile tokens for “arch”

Parameter:	{}profile-arch
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-arch := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-arch` are matched against the values in the `arch` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-audience

Profile tokens for “audience”

Parameter:	{}profile-audience
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-audience := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-audience` are matched against the values in the `audience` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-condition

Profile tokens for “condition”

Parameter:	{}profile-condition
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-condition := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-condition` are matched against the values in the `condition` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-conformance

Profile tokens for “conformance”

Parameter:	{}profile-conformance
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-conformance := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-conformance` are matched against the values in the `conformance` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-lang

Profile tokens for “xml:lang”

Parameter:	{}profile-lang
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-lang := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-lang` are matched against the values in the `xml:lang` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-os

Profile tokens for “os”

Parameter:	{}profile-os
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-os := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-os` are matched against the values in the `os` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-outputformat

Profile tokens for “outputformat”

Parameter:	{}profile-outputformat
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-outputformat := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-outputformat` are matched against the values in the `outputformat` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-revision

Profile tokens for “revision”

Parameter:	{}profile-revision
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-revision := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-revision` are matched against the values in the `revision` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-revisionflag

Profile tokens for “revisionflag”

Parameter:	{}profile-revisionflag
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-revisionflag := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-revisionflag` are matched against the values in the `revisionflag` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-role

Profile tokens for “role”

Parameter:	{}profile-role
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-role := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-role` are matched against the values in the `role` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-security

Profile tokens for “security”

Parameter:	{}profile-security
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-security := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-security` are matched against the values in the `security` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-separator

The profile separator character.

Parameter:	{}profile-separator
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-separator := ';'
```

## Description

The profiling parameters (and the profiling attributes) are strings. Profiling operates on sets of tokens. The strings in each case are divided into tokens by separating them at the `$profile-separator` character. Common values for separator include “;” and “ ” (space). See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-userlevel

Profile tokens for “userlevel”

Parameter:	{}profile-userlevel
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-userlevel := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-userlevel` are matched against the values in the `userlevel` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-vendor

Profile tokens for “vendor”

Parameter:	{}profile-vendor
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-vendor := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-vendor` are matched against the values in the `vendor` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$profile-wordsize

Profile tokens for “wordsize”

Parameter:	{}profile-wordsize
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$profile-wordsize := "
```

## Description

The tokens (separated by `$profile-separator`) specified for `$profile-wordsize` are matched against the values in the `wordsize` attribute. See *Section 2.7, “Effectivity attributes and profiling”*.

# \$qandadiv-default-toc

Generate a table-of-contents for `qandadiv` elements?

Parameter:	{ }qandadiv-default-toc
Defined in:	param.xsl
Used in:	param.xsl, modules/blocks.xsl

## Synopsis

```
$qandadiv-default-toc := $qandaset-default-toc
```

## Description

If this parameter is true, by default a table-of-contents will be generated at the beginning of each `qandadiv`.

# \$qandaset-default-label

The default label for `qandaset` elements.

Parameter{}qandaset-default-label

Defined inparam.xsl

Used in: param.xsl, modules/titles.xsl, modules/gentext.xsl, modules/xref.xsl

## Synopsis

```
$qandaset-default-label := 'number'
```

## Description

Question and answers can be rendered with a few different labeling styles. This parameter selects the default style.

# \$qandaset-default-toc

Generate a table-of-contents for `qandaset` elements?

Parameter:	{}qandaset-default-toc
Defined in:	param.xsl
Used in:	param.xsl, modules/blocks.xsl
Used by:	\$qandadiv-default-toc

## Synopsis

```
$qandaset-default-toc := 'true'
```

## Description

If this parameter is true, by default a table-of-contents will be generated at the beginning of each `qandaset`.

# \$refentry-generate-name

Use “Name” as the title of a `refentry` .

Parameter:	{}refentry-generate-name
Defined in:	param.xsl
Used in:	param.xsl, modules/refentry.xsl

## Synopsis

```
$refentry-generate-name := true()
```

## Description

If `$refentry-generate-name` is true, the title of the refentry page will be the word “Name”. Otherwise it will be the title of the subject of the reference page. Generally, exactly one of `$refentry-generate-name` or `$refentry-generate-title` should be true.

# \$refentry-generate-title

Use the subject of the page as the title of a `refentry` .

Parameter:	<code>{}refentry-generate-title</code>
Defined in:	<code>param.xsl</code>
Used in:	<code>param.xsl, modules/refentry.xsl</code>

## Synopsis

```
$refentry-generate-title := true()
```

## Description

If `$refentry-generate-title` is true, the title of the refentry page will be the subject of the page (the `refentrytitle`, `refdescriptor`, or the first `refname`). Generally, exactly one of `$refentry-generate-name` or `$refentry-generate-title` should be true.

# \$relax-ng-grammar

Validation grammar.

Parameter:	{}relax-ng-grammar
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$relax-ng-grammar as xs:string? := ()
```

## Description

If `$relax-ng-grammar` is provided, then the `$v:standard-transforms` will validate the document against the provided grammar.

# \$resource-base-uri

Base URI of additional resources.

Parameter: {}resource-base-uri

Defined in: param.xsl

Used in: main.xsl, param.xsl, modules/variable.xsl, modules/head.xsl

Used by: \$v:highlight-js-head-elements, \$v:prism-js-head-elements

## Synopsis

```
$resource-base-uri := './'
```

## Description

Web pages rely on additional resources (CSS, JavaScript, etc.) to render properly. The `$resource-base-uri` is used as the base URI for these resources.

# \$revhistory-style

Rendering style for `revhistory` elements.

Parameter: `{revhistory-style}`

Defined in: `param.xsl`

Used in: `param.xsl, modules/blocks.xsl`

## Synopsis

```
$revhistory-style := 'table'
```

## Description

The `revhistory` element can be rendered as a list or a table. If `$revhistory-style` is “table>”, it will be rendered using the tabular style, if it is “list”, it will be rendered using the list style.

*Example 1, “An example of char alignment”* shows an example of a `revhistory` element.

```
<revhistory>
<revision>
<revnumber>1.0.0</revnumber>
<date>2020-07-27</date>
<author>
<personname>Norman Tovey-Walsh</personname>
</author>
<revdescription>
<para>Version 1.0.0 released.</para>
</revdescription>
</revision>
<revision>
<revnumber>0.0.1</revnumber>
<date>2020-05-08T06:24:00Z</date>
<author>
<personname>Norman Tovey-Walsh</personname>
</author>
<revremark>Development begins.</revremark>
</revision>
</revhistory>
```

### Example 1. An example of char alignment

It is shown rendered as a table in *Figure 1, “An example of `revhistory` rendered as a table”*.

### Revision History

1.0.0	27 Jul 2020	Norman Tovey-Walsh	Version 1.0.0 released.
0.0.1	06:24 08 May 2020	Norman Tovey-Walsh	Development begins.

**Figure 1. An example of `revhistory` rendered as a table**

The same `revhistory` is shown rendered as a list in *Figure 2, “An example of `revhistory` rendered as a list”*.

### Revision History

- 1.0.0, 27 Jul 2020, Norman Tovey-Walsh  
Version 1.0.0 released.
- 0.0.1, 06:24 08 May 2020, Norman Tovey-Walsh  
Development begins.

**Figure 2. An example of `revhistory` rendered as a list**

The default style specified by `$revhistory-style` can be overridden on a case-by-case basis with the `db` processing instruction using the `revhistory-style` pseudo-attribute.

# \$section-numbers

Are sections numbered?

Parameter:	{}section-numbers
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl
Used by:	\$v:title-groups

## Synopsis

```
$section-numbers as xs:string := '1'
```

## Description

If `$section-numbers` is true, sections that do not have an explicit label will be labeled with their section number.

See *Section 3.5, “Controlling numeration”*.

# \$section-numbers-inherit

Include ancestor section labels?

## Synopsis

```
$section-numbers-inherit := 'true'
```

## Description

This parameter is no longer used. See *Section 3.5, “Controlling numeration”*.

# \$section-toc-depth

Depth of sections in the table of contents.

Parameter:	{}section-toc-depth
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl

## Synopsis

```
$section-toc-depth := 'unbounded'
```

## Description

When generating a Table of Contents, the `$section-toc-depth` determines the maximum depth of section to include. See also `$generate-toc` and `$generate-nested-toc`.

The `$section-toc-depth` should be a positive integer or the token `unbounded` to indicate arbitrary depth.

# \$sections-inherit-from

Identifies what inherited numeration applies to sections.

Parameter:	{}sections-inherit-from
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl

## Synopsis

```
$sections-inherit-from as xs:string := 'section'
```

## Description

If section numbers inherit from their ancestors, these are the ancestors they inherit from.

See *Section 3.5, “Controlling numeration”*.

# \$sections-number-from

Identifies where (top-level) section numeration begins.

Parameter:	{}sections-number-from
Defined in:	param.xsl
Used in:	param.xsl, modules/numbers.xsl

## Synopsis

```
$sections-number-from as xs:string := 'component'
```

## Description

Top-level section numbers (if sections are numbered) begin from here.

This parameter only applies to top-level sections. Nested sections are always numbered sequentially within their parent section.

See *Section 3.5, “Controlling numeration”*.

root

Top-level sections are numbered from the beginning of the document.

set

Top-level sections are numbered from their nearest ancestor set.

book

Top-level sections are numbered from their nearest ancestor book.

division

Top-level sections are numbered from their nearest ancestor division.

component

Top-level sections are numbered from their parent component.

# \$segmentedlist-style

Choose segmented list presentation style.

Parameter:	{}segmentedlist-style
Defined in:	param.xsl
Used in:	param.xsl, modules/lists.xsl

## Synopsis

```
$segmentedlist-style := 'table'
```

## Description

Segmented lists can be presented in two different ways: as lists or tables. The default is determined by `$segmentedlist-style` which must be either “`list`” or “`table`”.

Consider the following `segmentedlist` of (some of) the US states and their capitals:

```
1 <segmentedlist>
  <segtitle>State</segtitle>
  <segtitle>Capital</segtitle>
  <seglistitem>
5   <seg>Alabama</seg>
    <seg>Montgomery</seg>
  </seglistitem>
  <seglistitem>
    <seg>Alaska</seg>
10  <seg>Anchorage</seg>
  </seglistitem>
  <seglistitem>
    <seg>Arkansas</seg>
    <seg>Little Rock</seg>
15  </seglistitem>
</segmentedlist>
```

This can be rendered as a table, as shown in *Figure 1*, “Segmented list formatted as a table”:

State	Capital
Alabama	Montgomery
Alaska	Anchorage
Arkansas	Little Rock

**Figure 1. Segmented list formatted as a table**

Or as a list, as shown in *Figure 2*, “Segmented list formatted as a list”:

**State:** Alabama  
**Capital:** Montgomery  
**State:** Alaska  
**Capital:** Anchorage  
**State:** Arkansas  
**Capital:** Little Rock

**Figure 2. Segmented list formatted as a list**

The style can be selected on a per-list basis with the `db` processing instruction using the `segmentedlist-style` pseudo-attribute. A `table-summary` pseudo-attribute is also provided for the table summary in tabular presentations.

# \$sets-inherit-from

Identifies what inherited numeration applies to sets.

Parameter:	{}sets-inherit-from
Defined in:	param.xsl
Used in:	param.xsl, modules/titles.xsl

## Synopsis

```
$sets-inherit-from as xs:string := "
```

## Description

If set numbers inherit from their ancestors, these are the ancestors they inherit from.

See *Section 3.5, “Controlling numeration”*.

# \$sets-number-from

Identifies where set numeration begins.

Parameter:	{}sets-number-from
Defined in:	param.xsl
Used in:	param.xsl, modules/numbers.xsl

## Synopsis

```
$sets-number-from as xs:string := 'set'
```

## Description

Set numbers (if sets are numbered) begin from here.

See *Section 3.5, “Controlling numeration”*.

root

Sets are numbered from the beginning of the document.

set

Sets are numbered from their parent set.

# \$show-remarks

Show `remark` elements?

Parameter:	{}show-remarks
Defined in:	param.xsl
Used in:	param.xsl, modules/blocks.xsl

## Synopsis

```
$show-remarks := 'false'
```

## Description

The `remark` element is useful for editorial comments and other notes that are not intended to appear in the final publication. If `{}show-remarks` is true, then they will be included in the transformed result. Otherwise, they are omitted.

# \$sidebar-as-aside

Render `sidebar` as an `aside`?

Parameter: `{}sidebar-as-aside`

Defined in: `param.xsl`

Used in: `param.xsl, modules/blocks.xsl`

## Synopsis

```
$sidebar-as-aside := false()
```

## Description

The HTML `aside` element has specific semantics. Whether or not those semantics are consistent with the way `sidebar` is used in your DocBook documents is an open question. If `$sidebar-as-aside` is true, `sidebar` elements will be rendered as HTML `aside` elements, otherwise they will be rendered as `div` elements.

# \$sort-collation

Sorting collation.

Parameter:	{}sort-collation
Defined in:	param.xsl
Used in:	param.xsl, modules/glossary.xsl

## Synopsis

```
$sort-collation := 'http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive'
```

## Description

When items (the terms in a [glossary](#), for example) are sorted, this collation is used.

# \$table-accessibility

Accessibility settings for tables.

Parameter:	{table-accessibility}
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl

## Synopsis

```
$table-accessibility as xs:string := 'summary details'
```

## Description

The `$table-accessibility` parameter determines how accessibility features are added to CALS tables<sup>1</sup>.

The value of this parameter is a space-separated list of strings. If the list contains:

summary

A `summary` attribute will be added to the table if an `alt` element or a `textobject` containing a single `phrase` is available.

details

A `details` element will be added if there is a `textobject` containing anything other than a single `phrase` available.

The list may contain either or both values. If other values are present, they are ignored.

See *Example 1, “An example of table accessibility”*.

---

<sup>1</sup> This parameter does not apply to HTML tables because the HTML table model supports these features directly.

```

<table frame="all">
  <title>Square numbers</title>
  <alt>The first four whole numbers
and their squares.</alt>
  <textobject>
    <para>This table shows the first four whole numbers
and their squares. (The square of a number is that
number times itself.)</para>
  </textobject>
  <tgroup cols="2" colsep="1" rowsep="1">
    <thead>
      <row>
        <entry>
          <inlineequation>
            <mathphrase>n</mathphrase>
          </inlineequation>
        </entry>
        <entry>
          <inlineequation>
            <mathphrase>n<superscript>2</superscript>
          </mathphrase>
          </inlineequation>
        </entry>
      </row>
    </thead>
    <tbody>
      <row>
        <entry>1</entry>
        <entry>1</entry>
      </row>
      <row>
        <entry>2</entry>
        <entry>4</entry>
      </row>
      <row>
        <entry>3</entry>
        <entry>9</entry>
      </row>
      <row>
        <entry>4</entry>
        <entry>16</entry>
      </row>
    </tbody>
  </tgroup>

```

**Table 1. Square numbers**

n	$n^2$
1	1
2	4
3	9
4	16

Inspection of the HTML will reveal that the `summary` attribute is present on the element that wraps the table and a `details` element precedes it. These may or may not be rendered by your user agent depending on its accessibility features and settings.

See also [\\$mediaobject-accessibility](#).

# \$table-footnote-numeration

How are footnotes numbered in tables?

Parameter: {}table-footnote-numeration

Defined in: param.xsl

Used in: param.xsl, modules/footnotes.xsl, modules/chunk-cleanup.xsl

## Synopsis

```
$table-footnote-numeration := ('a')
```

## Description

There are different styles of footnote numeration. This parameter contains a list of the symbols that should be used to mark footnotes in tables. If there could be more footnotes than symbols in the list, the last symbol in the list must be a character suitable for formatting numbers. See [\\$footnote-numeration](#).

Footnote number begins with the first symbol in each table.

# \$theme-picker

Allow users to select themes?

Parameter:	{}theme-picker
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$theme-picker as xs:string := 'false'
```

## Description

If this parameter is true, a JavaScript “theme picker” will be included in the document. This is provided through a “hamburger menu” in the upper-right corner. Given that one of the ostensible goals of providing themes is accessibility, it’s somewhat ironic that this feature is not especially accessible. Suggestions welcome.

# \$transclusion-id-fixup

Transclusion ID fixup.

Parameter:	{}transclusion-id-fixup
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$transclusion-id-fixup as xs:string := 'none'
```

## Description

These stylesheets attempt to implement *Transclusion*. This parameter specifies the ID fixup.

# \$transclusion-link-scope

Transclusion link scope.

Parameter:	{}transclusion-link-scope
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$transclusion-link-scope as xs:string := 'global'
```

## Description

These stylesheets attempt to implement *Transclusion*. This parameter specifies the link scope.

# \$transclusion-prefix-separator

Transclusion separator.

Parameter:	{}transclusion-prefix-separator
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$transclusion-prefix-separator as xs:string := '---'
```

## Description

These stylesheets attempt to implement *Transclusion*. This parameter provides the separator for auto-generated prefixes.

# \$transclusion-suffix

Transclusion suffix.

Parameter:	{}transclusion-suffix
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$transclusion-suffix as xs:string := "
```

## Description

These stylesheets attempt to implement *Transclusion*. This parameter specifies the suffix used for transcluded IDs.

# \$transform-after

Transform the process document.

Parameter:	{}transform-after
Defined in:	param.xsl
Used in:	docbook.xsl, param.xsl
Used by:	t:docbook

## Synopsis

```
$transform-after := ()
```

## Description

Identifies the transform(s) that should be applied after the document has been transformed to HTML. See *Section 5.7, “The pre- and post-processing pipeline”*.

# \$transform-before

Transform the preprocessed document.

Parameter:	{}transform-before
Defined in:	param.xsl
Used in:	docbook.xsl, param.xsl

## Synopsis

```
$transform-before := ()
```

## Description

Identifies the transform(s) that should be applied before the document is transformation into HTML (but after the standard transformations). See *Section 5.7, “The pre- and post-processing pipeline”*.

# \$transform-original

Transform the original document.

Parameter:	{}transform-original
Defined in:	param.xsl
Used in:	docbook.xsl, param.xsl

## Synopsis

```
$transform-original := ()
```

## Description

Identifies the transform(s) that should be applied to the original document.  
See *Section 5.7, “The pre- and post-processing pipeline”*.

# \$unwrap-paragraphs

Attempt to unwrap paragraphs that contain block elements?

Parameter:	{}unwrap-paragraphs
Defined in:	param.xsl
Used in:	param.xsl, modules/blocks.xsl
Since:	2.0.13

## Synopsis

```
$unwrap-paragraphs as xs:string := 'false'
```

## Description

DocBook allows “block elements” (tables, figures, etc.) inside paragraphs or between paragraphs. The question of whether or not a table, figure, etc. is part of a paragraph is an editorial one. If you put the block element in the DocBook `para`, it will be in the resulting HTML `p` element.

Except, that’s not valid HTML. HTML deprives you of the editorial choice. If `$unwrap-paragraphs` is true, the stylesheets will attempt to unwrap block elements inside paragraphs, producing a sequence of paragraphs and blocks.

Note that with this option, a `p` element identified with an ID attribute is not guaranteed to contain all of the content that the DocBook paragraph contained.

# \$use-docbook-css

Create links to standard DocBook CSS stylesheets?

Parameter:	{}use-docbook-css
Defined in:	param.xsl
Used in:	param.xsl, modules/head.xsl

## Synopsis

```
$use-docbook-css as xs:string := 'true'
```

## Description

If this parameter is true, then the output will contain links to the standard DocBook CSS stylesheets:

```
<link href="${resource-base-uri}css/docbook.css"  
rel="stylesheet" media="screen"/>  
<link href="${resource-base-uri}css/docbook-paged.css"  
rel="stylesheet" media="print"/>
```

If `$use-minified-css` is true, then the links will be to minified CSS stylesheets. You can quickly disable all of these links by setting this parameter to false. There are a number of other stylesheets that are only conditionally included, see `$verbatim-syntax-highlight-css` and `$persistent-toc-css`. User defined stylesheets are added with `$user-css-links`.

# \$use-minified-css

Create links to minified CSS?

Parameter:	{}use-minified-css
Defined in:	param.xsl
Used in:	param.xsl, modules/head.xsl

## Synopsis

```
$use-minified-css as xs:string := 'false'
```

## Description

If this parameter is true, then the links to standard DocBook CSS stylesheets will be to minified versions instead of the “ordinary” versions. This does not effect links in `$user-css-links`.

# \$user-css-links

A list of user-defined CSS stylesheets.

Parameter:	{}user-css-links
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl

## Synopsis

```
$user-css-links := ()
```

## Description

This parameter is a space separated list of CSS stylesheets. Each will be added to the output after the standard stylesheets.

# \$variablelist-termlength-threshold

Threshold value for considering `varlistentry` terms “long”

Parameter: `{}variablelist-termlength-threshold`

Defined in: `param.xsl`

Used in: `param.xsl, modules/attributes.xsl`

## Synopsis

```
$variablelist-termlength-threshold := 20
```

## Description

If the sum of the lengths of the `term` elements in a `varlistentry` exceeds this threshold, the variable list will have a “`long`” class. This can be used in CSS or in other downstream processing to decide whether or not the terms and their corresponding list items should be aligned side-by-side.

The length computation is just the number of characters. No attempt is made to adjust for wide or narrow characters.

# \$verbatim-callouts

A list determining how callouts are processed.

Parameter:	{}verbatim-callouts
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:verbatim-callouts

## Synopsis

\$verbatim-callouts as xs:string := 'linecolumn lines lineranges-first'

## Description

The `$verbatim-callouts` parameter is a space separated list of token values. These token values determine which kinds of callouts in a `programlistingco` or `screenco` will be processed and how.

linecolumn

If “`linecolumn`” appears in the list then callouts that have a `units` attribute of `linecolumn` and specify both a line and column will be processed. A callout marker will be placed on the line and before the column indicated. Additional lines and columns will be added to the listing if necessary.

lines

If “`lines`” appears in the list then callouts that have a `units` attribute of `linecolumn` and specify only a line will be processed. A callout marker will be placed on the line and before the `$callout-default-column`.

lineranges-first

If “`lineranges-first`” appears in the list then callouts that have a `units` attribute of `linerange` will be processed. A callout marker will be placed on the first line and before the `$callout-default-column`.

Only one of `lineranges-first` and `lineranges-all` should be specified.

## lineranges-all

If “`lineranges-first`” appears in the list then callouts that have a `units` attribute of `linerange` will be processed. A callout marker will be placed on every line in the range before the `$callout-default-column`.

Only one of `lineranges-first` and `lineranges-all` should be specified.

There is no support for `area`s with a `units` value of “`calspair`” or “`linecolumnpair`”. They are always ignored.

# \$verbatim-line-style

List of verbatim elements to be rendered in the line style.

Parameter:	{}verbatim-line-style
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:verbatim-line-style

## Synopsis

```
$verbatim-line-style := 'programlisting programlistingco  
screen screenco synopsis'
```

## Description

This parameter is a space-separated list of element names (local names). Verbatim elements that appear in this list will be formatted using the line style by default. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

# \$verbatim-number-every-nth

Line numbering frequency.

Parameter:	{}verbatim-number-every-nth
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:verbatim-number-every-nth

## Synopsis

```
$verbatim-number-every-nth := 5
```

## Description

When formatting verbatim environments with line numbers, every `$verbatim-number-every-nth` line is numbered.

# \$verbatim-number-first-line

Number the first line?

Parameter:	{}verbatim-number-first-line
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:verbatim-number-first-line

## Synopsis

```
$verbatim-number-first-line := 'true'
```

## Description

When formatting verbatim environments with line numbers, if `$verbatim-number-first-line` line is true, the first line will be numbered even it isn't one of the `$verbatim-number-every-nth` lines.

# \$verbatim-number-minlines

Shortest listing to number.

Parameter:	{}verbatim-number-minlines
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:verbatim-number-minlines

## Synopsis

```
$verbatim-number-minlines := '5'
```

## Description

When formatting verbatim environments with line numbers, environments less than `$verbatim-number-minlines` in length will not be numbered.

# \$verbatim-number-separator

Separator between line numbers and lines.

Parameter:	{}verbatim-number-separator
Defined in:	param.xsl
Used in:	param.xsl, modules/verbatim.xsl

## Synopsis

```
$verbatim-number-separator := '|'
```

## Description

When formatting verbatim environments with line numbers, this separator is inserted (wrapped in a `span` with a `class` of `nsep`) between the line number and the text of the line.

In most presentations, CSS prevents the `nsep` from being displayed; instead, a CSS border is used as a separator. The `$verbatim-number-separator` is most often seen with text-only browsers.

# \$verbatim-numbered-elements

Verbatim environments that should be numbered.

Parameter:	{}verbatim-numbered-elements
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:verbatim-numbered-elements

## Synopsis

```
$verbatim-numbered-elements := 'programlisting programlistingco'
```

## Description

This parameter is a space-separated list of element names (local names). Verbatim elements that appear in this list will be formatted with line numbers. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

# \$verbatim-plain-style

List of verbatim elements to be rendered in the plain style.

Parameter:	{}verbatim-plain-style
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:verbatim-plain-style

## Synopsis

```
$verbatim-plain-style as xs:string := 'address literallayout funcsynopsisinfo classsynopsisinfo'
```

## Description

This parameter is a space-separated list of element names (local names). Verbatim elements that appear in this list will be formatted using the plain style by default. For a discussion of verbatim elements and styles, see [Section 5.3, “Verbatim styles”](#).

See also [\\$v:verbatim-plain-style](#).

# \$verbatim-space

The space character to use when padding verbatim lines.

Parameter:     {}verbatim-space

Defined in:    param.xsl

Used in:       param.xsl, modules/variable.xsl

Used by:       \$v:verbatim-space

## Synopsis

```
$verbatim-space := ''
```

## Description

When a verbatim line has to be padded in order to get a callout to appear in the correct column, this character will be used as the padding character. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

# \$verbatim-style-default

The default verbatim style.

Parameter:	{}verbatim-style-default
Defined in:	param.xsl
Used in:	param.xsl, modules/verbatim.xsl

## Synopsis

```
$verbatim-style-default := 'lines'
```

## Description

The default verbatim style for verbatim environments that do not specify a style explicitly. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

# \$verbatim-syntax-highlight-css

Stylesheet for syntax highlighting.

Parameter:     {}verbatim-syntax-highlight-css

Defined in:    param.xsl

Used in:       param.xsl, modules/head.xsl

## Synopsis

```
$verbatim-syntax-highlight-css := 'css/pygments.css'
```

## Description

If the `$verbatim-syntax-highlight-languages` parameter is non-empty, then this stylesheet will be linked from the HTML document. The purpose of this stylesheet is to provide styling (fonts, colors, etc.) for syntax highlighted verbatim listings.

# \$verbatim-syntax-highlight-languages

Languages for which syntax highlighting should be performed.

Parameter: `{}verbatim-syntax-highlight-languages`

Defined in: `param.xsl`

Used in: `param.xsl, modules/variable.xsl`

Used by: `$v:verbatim-syntax-highlight-languages`

## Synopsis

```
$verbatim-syntax-highlight-languages := 'python perl html xml xslt xquery javascript json'
```

## Description

This parameter is a space-separated list of programming languages. If the Pygments (<https://pygments.org/>) syntax highlighter is available, the stylesheets can apply syntax highlighting to program listings. If the program listing language (as specified in the `language` attribute) appears in the list of `$verbatim-syntax-highlight-languages`, syntax highlighting will be attempted.

### Note



Syntax highlighting is incompatible with inline markup in the program listing; the embedded markup will be ignored.

The examples in this guide show syntax highlighting applied to XML.

See also `$v:verbatim-syntax-highlight-options` and  
`$v:verbatim-syntax-highlight-pygments-options`.

# \$verbatim-syntax-highlighter

Selects the syntax highlighter.

Parameter: {} verbatim-syntax-highlighter

Defined in: param.xsl

Used in: param.xsl, modules/head.xsl, modules/verbatim.xsl

Used by: f:verbatim-style(), f:verbatim-highlight(), f:verbatim-numbered()

## Synopsis

```
$verbatim-syntax-highlighter as xs:string := 'pygments'
```

## Description

This parameter specifies the syntax highlighter to use. Most of the parameters related to syntax highlighting only work if the “ pygments ” highlighter is selected. However, that requires running an external process which might not be available and which, even if it is available, has an impact on performance. Using Pygments has the advantage that the stylesheets have more control over the process and it is available both for online and paged media formats.

The alternative to running an external program is to use JavaScript to highlight the listing in the browser. This is faster at formatting time, but limits syntax highlighting to those environments where JavaScript is available.

The following options are supported:

pygments

This is the default highlighter.

highlight.js

Uses the *highlight.js* JavaScript library to perform syntax highlighting. The `$highlight-js-head-elements` variable determines what CSS stylesheets and JavaScript libraries are loaded for this option.

## prism

Uses the *Prism* JavaScript library to perform syntax highlighting. The `$prism-js-head-elements` variable determines what CSS stylesheets and JavaScript libraries are loaded for this option.

## none

Disables syntax highlighting but also suppresses messages about features that are not available because syntax highlighting is disabled.

# \$verbatim-table-style

List of verbatim elements to be rendered in the table style.

Parameter:	{}verbatim-table-style
Defined in:	param.xsl
Used in:	param.xsl, modules/variable.xsl
Used by:	\$v:verbatim-table-style

## Synopsis

```
$verbatim-table-style := "
```

## Description

This parameter is a space-separated list of element names (local names). Verbatim elements that appear in this list will be formatted using the table style by default. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

# \$verbatim-trim-trailing-blank-lines

Trim trailing blank lines off verbatim environments?

Parameter:	{}verbatim-trim-trailing-blank-lines
Defined in:	param.xsl
Used in:	param.xsl, modules/verbatim.xsl
Used by:	f:verbatim-trim-trailing()

## Synopsis

```
$verbatim-trim-trailing-blank-lines := 'true'
```

## Description

Trailing blank lines can be introduced into verbatim environments in a number of ways. Transcluded text files may have trailing blank lines, for example, and authors often put a newline before the closing tag of an environment.

This can produce spurious looking listings in the documentation. If trailing newlines aren't significant, setting `$verbatim-trim-trailing-blank-lines` will cause the stylesheets to trim blank (that is, entirely empty) lines from the end of verbatim environments.

# \$warn-about-missing-localizations

Warn about missing localization?

Parameter:     {}warn-about-missing-localizations

Defined in:    param.xsl

Used in:       param.xsl, modules/gentext.xsl

## Synopsis

```
$warn-about-missing-localizations as xs:string := 'true'
```

## Description

If this parameter is true, then warning messages will be issued for missing localizations.

# \$xlink-arclist-after

Suffix used for inline rendering of a list of XLink arcs.

Parameter:	{}xlink-arclist-after
Defined in:	param.xsl
Used in:	param.xsl, modules/xlink.xsl

## Synopsis

```
$xlink-arclist-after := ']' '
```

## Description

When rendering a list of XLinks inline, this precedes the list.

# \$xlink-arclist-before

Prefix used for inline rendering of a list of XLink arcs.

Parameter:	{}xlink-arclist-before
Defined in:	param.xsl
Used in:	param.xsl, modules/xlink.xsl

## Synopsis

```
$xlink-arclist-before := '['
```

## Description

When rendering a list of XLinks inline, this precedes the list.

# \$xlink-arclist-sep

Link separator used when rendering a list of XLink arcs.

Parameter:	{}xlink-arclist-sep
Defined in:	param.xsl
Used in:	param.xsl, modules/xlink.xsl

## Synopsis

```
$xlink-arclist-sep := ','
```

## Description

When rendering a list of XLinks inline, this separates the links.

# \$xlink-arclist-titlesep

Title separator used when rendering a list of XLink arcs.

Parameter:	{}xlink-arclist-titlesep
Defined in:	param.xsl
Used in:	param.xsl, modules/xlink.xsl

## Synopsis

```
$xlink-arclist-titlesep := ':'
```

## Description

When rendering a list of XLinks inline, this separates the link titles.

# \$xlink-icon-closed

Icon indicating hidden XLink links.

Parameter:	{}xlink-icon-closed
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$xlink-icon-closed := ()
```

## Description

If the JavaScript presentation of extended XLinks is used, this icon marks the place where a multi-ended link can be revealed. The default icon is “▶” (&#x25B6;). See also: [\\$xlink-icon-open](#).

# \$xlink-icon-open

XLink CSS.

Parameter:	{}xlink-icon-open
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$xlink-icon-open := ()
```

## Description

If the JavaScript presentation of extended XLinks is used, this icon marks the place where a multi-ended link has been revealed. The default icon is “▼” (&#x25BC;).

### Note



The default icons are quite large compared to running text of the same size. To compensate, they're styled at a `font-size` of 70%. If you select different icons, you may need to adjust the `font-size` associated with the `xlink-arc-list` class.

# \$xlink-js

Script to support extended XLinks.

Parameter:	{}xlink-js
Defined in:	param.xsl
Used in:	main.xsl, param.xsl

## Synopsis

```
$xlink-js := 'js/xlink.js'
```

## Description

If the extended XLinks appear in the document, a link to this script will be added to the document. This script must contain the JavaScript necessary to support rendering extended XLinks.

# \$xlink-style

Presentation style for XLink extended links.

Parameter:	{}xlink-style
Defined in:	param.xsl
Used in:	param.xsl, modules/xlink.xsl
Used by:	f:xlink-style()

## Synopsis

```
$xlink-style := 'document'
```

## Description

The stylesheets support XLink extended links. Extended links make it possible to have one-to-many links. That is, a single link such as *DocBook* [[DocBook.org](https://docbook.org/) (<https://docbook.org/>)], *DocBook* on Wikipedia (<https://en.wikipedia.org/wiki/DocBook>) may point to more than one target. Three styles are supported:

inline

Renders a link to each target inline after the originating link.

javascript

Uses JavaScript to render the links in a drop-down menu.

document

The preceding XLink styles apply to the entire document, you cannot specify inline presentation in some cases and JavaScript presentation in others. However, if you specify “`document`” as the value for `$xlink-style`, then a `db` processing instruction in the document’s top-level `info` element can be used to control the presentation. Specify the style in the `xlink-style` pseudo-attribute.

If the `document` style is selected and no processing instruction specifying a style exists, the `$xlink-style-default` style will be selected.

# \$xlink-style-default

Default XLink style.

Parameter:	{}xlink-style-default
Defined in:	param.xsl
Used in:	param.xsl, modules/xlink.xsl
Used by:	f:xlink-style()

## Synopsis

```
$xlink-style-default := 'inline'
```

## Description

Specifies the default XLink style if “document” is specified for the \$xlink-style and no processing instruction specifying a style exists.

Must be either “inline” or “javascript”.

# \$xspec

Are we running XSpec tests?

Parameter:	{}xspec
Defined in:	param.xsl
Used in:	param.xsl

## Synopsis

```
$xspec as xs:string := 'false'
```

## Description

This parameter is only true if the stylesheets are running in the context of the XSpec test harness. This allows tests to be crafted such that normal variations (Saxon HE vs. EE, the version of the stylesheets, or the current time, for example) don't introduce spurious differences in the results, causing tests to fail.



## II. Variables reference

# \$arg-choice-def-close-str, ...

Punctuation marks used in formatting [cmdsynopsis](#).

Variable: {<http://docbook.org/ns/docbook/variables>}arg-choice-def-close-str  
{<http://docbook.org/ns/docbook/variables>}arg-choice-def-open-str  
{<http://docbook.org/ns/docbook/variables>}arg-choice-opt-close-str  
{<http://docbook.org/ns/docbook/variables>}arg-choice-opt-open-str  
{<http://docbook.org/ns/docbook/variables>}arg-choice-plain-close-str  
{<http://docbook.org/ns/docbook/variables>}arg-choice-plain-open-str  
{<http://docbook.org/ns/docbook/variables>}arg-choice-req-close-str  
{<http://docbook.org/ns/docbook/variables>}arg-choice-req-open-str  
{<http://docbook.org/ns/docbook/variables>}arg-or-sep  
{<http://docbook.org/ns/docbook/variables>}arg-rep-def-str  
{<http://docbook.org/ns/docbook/variables>}arg-rep-norepeat-str  
{<http://docbook.org/ns/docbook/variables>}arg-rep-repeat-str

Defined inmodules/variable.xsl (12)

Used in: modules/programming.xsl

## Synopsis

```
<xsl:variable name="v:arg-choice-opt-open-str">
  <span class="cmdpunct">[</span>
</xsl:variable>
```

```
<xsl:variable name="v:arg-choice-opt-close-str">
  <span class="cmdpunct">]</span>
</xsl:variable>
```

```
<xsl:variable name="v:arg-choice-req-open-str">
  <span class="cmdpunct">{</span>
</xsl:variable>
```

```
<xsl:variable name="v:arg-choice-req-close-str">  
  <span class="cmdpunct">}</span>  
</xsl:variable>
```

```
<xsl:variable name="v:arg-choice-plain-open-str">  
  <xsl:text/>  
</xsl:variable>
```

```
<xsl:variable name="v:arg-choice-plain-close-str">  
  <xsl:text/>  
</xsl:variable>
```

```
<xsl:variable name="v:arg-choice-def-open-str">  
  <span class="cmdpunct">[</span>  
</xsl:variable>
```

```
<xsl:variable name="v:arg-choice-def-close-str">  
  <span class="cmdpunct">]</span>  
</xsl:variable>
```

```
<xsl:variable name="v:arg-rep-repeat-str">  
  <span class="cmdpunct">...</span>  
</xsl:variable>
```

```
<xsl:variable name="v:arg-rep-norepeat-str">  
  <xsl:text/>  
</xsl:variable>
```

```
<xsl:variable name="v:arg-rep-def-str">  
  <xsl:text/>  
</xsl:variable>
```

```
<xsl:variable name="v:arg-or-sep">  
  <span class="cmdpunct"> | </span>  
</xsl:variable>
```

# Description

The `cmdsynopsis` element is used to model the syntax of command line interfaces. Broadly, a command line interface consists of a command followed by a variety of options. These options can be optional or required and may or may not be repeatable.

User expectations about how these should be formatted vary by platform and publisher. The `$v:arg-...` variables are designed to make it easy to adapt to many common forms of presentation. Each contains (usually) a single character used as a delimiter:

Variable	Use in a synopsis	Default
<code>\$v:arg-choice-def-close-str</code>	Follows a default option	]
<code>\$v:arg-choice-def-open-str</code>	Precedes a default option	[
<code>\$v:arg-choice-opt-close-str</code>	Follows an optional option	]
<code>\$v:arg-choice-opt-open-str</code>	Precedes an optional option	[
<code>\$v:arg-choice-plain-close-str</code>	Follows an option identified as “plain”	(empty string)
<code>\$v:arg-choice-plain-open-str</code>	Precedes an option identified as “plain”	(empty string)
<code>\$v:arg-choice-req-close-str</code>	Follows a required option	}
<code>\$v:arg-choice-req-open-str</code>	Precedes a required option	{
<code>\$v:arg-or-sep</code>	Separator between options in an “or” group	
<code>\$v:arg-rep-def-str</code>	Identifies a repeatable default option	(empty string)
<code>\$v:arg-rep-norepeat-str</code>	Identifies a non-repeatable option	(empty string)
<code>\$v:arg-rep-repeat-str</code>	Identifies an explicitly repeatable option	...

<sup>1</sup>

A contrived `cmdsynopsis` appears in *Example 1, “An example of a cmdsynopsis”*.

---

<sup>1</sup> Each of the default characters is placed in an HTML `span` with the class “`cmdpunct`”.

```
<cmdsynopsis>
  <command>command</command>
  <arg choice="plain">
    <option>--path <replaceable>PATH</replaceable>
  </option>
  </arg>
  <arg>-opt</arg>
  <arg choice="req">-req</arg>
  <arg rep="repeat">-D<replaceable>name</replaceable>
  </arg>
  <group>
    <arg choice="plain">-M</arg>
    <arg choice="plain">-MD</arg>
  </group>
</cmdsynopsis>
```

**Example 1. An example of a cmdsynopsis**

One possible rendering of such a synopsis:

command --path *PATH* [-opt] {-req} [-D*name*...] [-M | -MD]

# \$err:DYNAMIC-PROFILE-EVAL-ERROR

Error evaluating dynamic profile expression.

Variable: {http://docbook.org/ns/docbook/errors}DYNAMIC-PROFILE-EVAL-ERROR

Defined imodules/errors.xsl

Used in: Not used.

## Synopsis

`$dbe:DYNAMIC-PROFILE-EVAL-ERROR := xs:QName('dbe:DYNAMIC-PROFILE-EVAL-ERROR')`

## Description

This error is raised if the variable referenced in a dynamic profile expression (see *Section 2.7, “Effectivity attributes and profiling”*) is not available and the `$dynamic-profile-error` configuration is “`error`”.

# \$err:DYNAMIC-PROFILE-SYNTAX-ERROR

Internal error processing dynamic profiles.

Variable: {<http://docbook.org/ns/docbook/errors>}DYNAMIC-PROFILE-SYNTAX-ERROR

Defined in: modules/errors.xsl

Used in: Not used.

## Synopsis

`$dbe:DYNAMIC-PROFILE-SYNTAX-ERROR := xs:QName('dbe:DYNAMIC-PROFILE-SYNTAX-E`

## Description

This is an internal error. It indicates that the syntax of a dynamic profiling expression was unrecognized. Since dynamic profiling should only be applied to expressions that *have* been recognized, this error should never happen.

# \$err:INTERNAL-HIGHLIGHT-ERROR

Internal error in syntax highlighting.

Variable: {http://docbook.org/ns/docbook/errors}INTERNAL-HIGHLIGHT-ERROR

Defined in modules/errors.xsl

Used in: highlight.xsl

## Synopsis

`$dbe:INTERNAL-HIGHLIGHT-ERROR := xs:QName('dbe:INTERNAL-HIGHLIGHT-ERROR')`

## Description

This is an internal error that should never happen. It indicates that stylesheet processing of program listings attempted to use the syntax highlighter when the syntax highlighter was known to be unavailable. If you see this error, please file an issue (<https://github.com/docbook/xsltNG/issues>) for it.

# \$err:INTERNAL-RENUMBER-ERROR

Internal error in chunk formatting.

Variable: {http://docbook.org/ns/docbook/errors}INTERNAL-RENUMBER-ERROR

Defined in modules/errors.xsl

Used in: modules/chunk-cleanup.xsl

## Synopsis

```
$dbe:INTERNAL-RENUMBER-ERROR := xs:QName('dbe:INTERNAL-RENUMBER-ERROR')
```

## Description

This is an internal error that should never happen. It indicates that stylesheet processing of footnotes during “chunking” has gone awry. If you see this error, please file an issue (<https://github.com/docbook/xsltNG/issues>) for it.

# \$err:INVALID-AREAREFS

Callout area refers to invalid target.

Variable: {http://docbook.org/ns/docbook/errors}INVALID-AREAREFS

Defined in: modules/errors.xsl

Used in: modules/lists.xsl

## Synopsis

```
$dbe:INVALID-AREAREFS := xs:QName('dbe:INVALID-AREAREFS')
```

## Description

This error indicates that a `calloutlist` includes an ID/IDREF link to a callout, but the element identified by the ID is not an `areaset`, `area`, or `callout`.

This should (<https://github.com/docbook/docbook/issues/151>) be identified as a validation error by a Schematron rule.

# \$err:INVALID-CALS

CALS table is invalid.

Variable:	{http://docbook.org/ns/docbook/errors}INVALID-CALS
Defined in:	modules/errors.xsl
Used in:	modules/tablecals.xsl

## Synopsis

```
$dbe:INVALID-CALS := xs:QName('dbe:INVALID-CALS')
```

## Description

This error is raised if a CALS table is invalid. The following error conditions give rise to this error:

- The number of columns in the table exceeds the specified `cols` value on `tgroup`.
- There is a named reference to a `colspec` but no `colspec` with that name exists.
- There is a named reference to a `spanspec` but no `spanspec` with that name exists.

# \$err:INVALID-CONSTRAINT

Incorrect constraint reference.

Variable: {http://docbook.org/ns/docbook/errors}INVALID-CONSTRAINT

Defined in: modules/errors.xsl

Used in: modules/programming.xsl

## Synopsis

```
$dbe:INVALID-CONSTRAINT := xs:QName('dbe:INVALID-CONSTRAINT')
```

## Description

This error is raised if a `constraint` link points to something other than a `constraintdef`.

This should (<https://github.com/docbook/docbook/issues/152>) be identified as a validation error by a Schematron rule.

# \$err:INVALID-DYNAMIC-PROFILE-ERROR

Invalid value specified for `$dynamic-profile-error`.

Variable: {<http://docbook.org/ns/docbook/errors>}INVALID-DYNAMIC-PROFILE-ERROR

Defined in `modules/errors.xsl`

Used in: Not used.

## Synopsis

`$dbe:INVALID-DYNAMIC-PROFILE-ERROR := xs:QName('dbe:INVALID-DYNAMIC-PROFILE-E`

## Description

This error is raised if the value of `$dynamic-profile-error` is not a recognized value.

# \$err:INVALID-INJECT

Invalid callout area specification.

Variable: {http://docbook.org/ns/docbook/errors}INVALID-INJECT

Defined in: modules/errors.xsl

Used in: modules/verbatim.xsl

## Synopsis

```
$dbe:INVALID-INJECT := xs:QName('dbe:INVALID-INJECT')
```

## Description

This error is raised if a `programlistingco` or `screenco` contains invalid areas. This can arise from invalid markup, unsupported units in an area, or unparseable coordinates in an area.

# \$err:INVALID-NAME-STYLE

Invalid name style.

Variable: {http://docbook.org/ns/docbook/errors}INVALID-NAME-STYLE

Defined in: modules/errors.xsl

Used in: modules/info.xsl

Used by: t:person-name

## Synopsis

```
$dbe:INVALID-NAME-STYLE := xs:QName('dbe:INVALID-NAME-STYLE')
```

## Description

This error indicates that a request was made to format a personal name (see `t:person-name`) with an unknown style name.

# \$err:INVALID-PRODUCTIONRECAP

Incorrect production reference.

Variable: {http://docbook.org/ns/docbook/errors}INVALID-PRODUCTIONRECAP

Defined in modules/errors.xsl

Used in: modules/programming.xsl

## Synopsis

`$dbe:INVALID-PRODUCTIONRECAP := xs:QName('dbe:INVALID-PRODUCTIONRECAP')`

## Description

This error is raised if a `productionrecap` link points to something other than a `production`.

This should (<https://github.com/docbook/docbook/issues/152>) be identified as a validation error by a Schematron rule.

# \$err:INVALID-RESULTS-REQUESTED

Invalid result form parameter.

Variable: {http://docbook.org/ns/docbook/errors}INVALID-RESULTS-REQUESTED

Defined in modules/errors.xsl

Used in: docbook.xsl

Used by: t:docbook

## Synopsis

```
$dbe:INVALID-RESULTS-REQUESTED := xs:QName('dbe:INVALID-RESULTS-REQUESTED')
```

## Description

This error is raised if the `$result` parameter passed to `t:docbook` is not one of “raw-results”, “chunked-results”, or “main-document”.

# \$err:INVALID-TEMPLATE

Invalid template reference.

Variable: {http://docbook.org/ns/docbook/errors}INVALID-TEMPLATE

Defined in: modules/errors.xsl

Used in: modules/templates.xsl

## Synopsis

```
$dbe:INVALID-TEMPLATE := xs:QName('dbe:INVALID-TEMPLATE')
```

## Description

This error is raised if a template reference names a template that does not exist.

# \$err:INVALID-TRANSFORM

Error processing a transform.

Variable:	{http://docbook.org/ns/docbook/errors}INVALID-TRANSFORM
Defined in:	modules/errors.xsl
Used in:	docbook.xsl
Used by:	t:docbook

## Synopsis

```
$dbe:INVALID-TRANSFORM := xs:QName('dbe:INVALID-TRANSFORM')
```

## Description

The lists of transforms in `$transform-original`, `$transform-before`, and `$transform-after` must either be strings (the stylesheet location) or a map that specifies the stylesheet location and other properties.

This error is raised if an element of the list is neither a map nor a string.

# \$v:personal-name-styles

The list of known personal name styles.

Variable: {http://docbook.org/ns/docbook/variables}personal-name-styles

Defined in: modules/variable.xsl

Used in: modules/info.xsl

## Synopsis

```
$v:personal-name-styles := ('first-last', 'last-first', 'FAMILY-given')
```

## Description

The stylesheets can format personal names in a variety of ways. This variable contains the list of known ways, see [t:person-name](#).

# \$v:VERSION

The stylesheet version.

Variable: {http://docbook.org/ns/docbook/variables}VERSION

Defined in: VERSION.xsl

Used in: modules/head.xsl, modules/inlines.xsl

## Synopsis

```
$v:VERSION := '2.1.0'
```

## Description

This variable holds the version number of the stylesheets.

# \$v:VERSION-ID

A unique version identifier.

Variable: {http://docbook.org/ns/docbook/variables}VERSION-ID

Defined in: VERSION.xsl

Used in: modules/head.xsl

## Synopsis

```
$v:VERSION-ID := 'SNAPSHOT'
```

## Description

Generally speaking the `$VERSION` is sufficient to identify the stylesheets. The `$VERSION-ID` adds a unique identifier derived from the latest git (<https://en.wikipedia.org/wiki/Git>) commit.

# \$v:admonition-icons

Admonition icons.

Variable: {http://docbook.org/ns/docbook/variables}admonition-icons

Defined in: modules/variable.xsl

Used in: modules/admonitions.xsl

## Synopsis

```
<xsl:variable name="v:admonition-icons">
  <tip>☞</tip>
  <note>①</note>
  <important>☝</important>
  <caution>⚠</caution>
  <warning>⚠</warning>
  <danger>⚡</danger>
</xsl:variable>
```

## Description

These are the icons that will be presented next to admonitions. They're single Unicode characters in the default distribution, but they can be replaced by graphics or other markup.

# \$v:annotation-close

Annotation close button.

Variable: {http://docbook.org/ns/docbook/variables}annotation-close

Defined in: modules/variable.xsl

Used in: main.xsl

## Synopsis

```
<xsl:variable name="v:annotation-close"
    as="element()">
    <span>X</span>
</xsl:variable>
```

## Description

The contents of this variable will be rendered as the “close” icon for popup annotations.

# \$v:as-json

Map for serializing JSON.

Variable: {http://docbook.org/ns/docbook/variables}as-json

Defined in: modules/variable.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport(), f:object-properties()

## Synopsis

```
$v:as-json := map {'method':'json','indent':true()}
```

## Description

This is a convenience variable for serializing maps as JSON. Used mostly in debugging.

# \$v:as-xml

Map for serializing XML.

Variable: {http://docbook.org/ns/docbook/variables}as-xml

Defined in: modules/variable.xsl

Used in: modules/objects.xsl

## Synopsis

```
$v:as-xml := map {'method':'xml','indent':true()}
```

## Description

This is a convenience variable for serializing maps as JSON. Used mostly in debugging.

# \$v:bridgehead-map

Maps `bridgehead` `renderas` values.

Variable: `{http://docbook.org/ns/docbook/variables}bridgehead-map`

Defined in: `modules/sections.xsl`

Used in: `modules/sections.xsl`

## Synopsis

```
<xsl:variable name="v:bridgehead-map"
    as="map(*)">
    <xsl:map>
        <xsl:map-entry key="sect1" select="h2"/>
        <xsl:map-entry key="sect2" select="h3"/>
        <xsl:map-entry key="sect3" select="h4"/>
        <xsl:map-entry key="sect4" select="h5"/>
        <xsl:map-entry key="sect5" select="h5"/>
        <xsl:map-entry key="sect6" select="h5"/>
        <xsl:map-entry key="block" select="div"/>
    </xsl:map>
</xsl:variable>
```

## Description

The `bridgehead` element allows the author to insert a heading without regard to the logical structure of the document. This variable controls how the values provided in the `renderas` attribute map to HTML. If there is no mapping for the specified rendering, it is rendered as a `div`.

# \$v:chunk

Are we chunking on this run?

Variable: {http://docbook.org/ns/docbook/variables}chunk

Defined in modules/variable.xsl

Used in: param.xsl, modules/variable.xsl, modules/chunk.xsl, modules/chunk-cleanup.xsl, modules/chunk-output.xsl

Used by: \$chunk-output-base-uri, f:chunk()

## Synopsis

```
$v:chunk as xs:boolean := not(normalize-space($chunk) = "")
```

## Description

If `$v:chunk` is true, then the stylesheets are producing chunked output. See *Section 2.6, ““Chunked” output”*.

# \$v:chunk-filter-namespaces

Namespace context for chunking expressions.

Variable:	{http://docbook.org/ns/docbook/variables}chunk-filter-namespaces
Defined in	modules/variable.xsl
Used in:	modules/chunk.xsl

## Synopsis

```
<xsl:variable name="v:chunk-filter-namespaces"
    as="namespace-node()*">
    <xsl:namespace name="db" select=""http://docbook.org/ns/docbook""/>
</xsl:variable>
```

## Description

When the `$chunk-include` and `$chunk-exclude` expressions are evaluated, the namespace bindings in this variable will be in-scope.

# \$v:chunk-renumber-footnotes

Renumber footnotes when chunking?

Variable: {http://docbook.org/ns/docbook/variables}chunk-renumber-footnotes

Defined in modules/variable.xsl

Used in: modules/chunk-cleanup.xsl

## Synopsis

```
$v:chunk-renumber-footnotes as xs:boolean := f:is-true($chunk-renumber-footnotes)
```

## Description

Footnotes are generally numbered (or otherwise marked, see `$footnote-numeration`) sequentially throughout a document. If the document is being broken into chunks, it may seem odd if the only footnote in a chunk is labeled “5”. If `$v:chunk-renumber-footnotes` is true, the stylesheets will attempt to renumber footnotes in each chunk so that they begin at the first mark in each chunk.

### Note



If your document uses `footnoteref` and the reference and the footnote are in different chunks, this may lead to very confusing numeration!

# \$v:custom-localizations

Customize localizations.

Variable:	{http://docbook.org/ns/docbook/variables}custom-localizations
Defined in:	modules/variable.xsl
Used in:	modules/gentext.xsl
Since:	2.0.0

## Synopsis

`$v:custom-localizations as document-node()? := ()`

## Description

Provides overrides for localization. See *Chapter 4, Localization*.

# \$v:debug

Debugging flags.

Variable: {http://docbook.org/ns/docbook/variables}debug

Defined in modules/variable.xsl

Used in: modules/variable.xsl, modules/titlepage.xsl, modules/objects.xsl, modules/chunk-cleanup.xsl, modules/chunk-output.xsl

Static: Yes

## Synopsis

```
$v:debug as xs:string* := tokenize($debug, '[,\s]+') ! normalize-space(.)
```

## Description

The `$v:debug` variable contains a sequence of debugging flags. These are derived from tokenizing the `$debug` parameter value.

# \$v:formal-object-title-placement

Placement of formal object titles.

Variable: {http://docbook.org/ns/docbook/variables}formal-object-title-placement

Defined in modules/variable.xsl

Used in: modules/blocks.xsl, modules/tablecals.xsl

## Synopsis

`$v:formal-object-title-placement as map(xs:string, xs:string) := fp:parse-key-value-pairs( tokenize($fc...`

## Description

This variable, usually derived from `$formal-object-title-placement` is a map from element (local) names to placements. The placement is either “`before`” or “`after`”.

# \$v:formalgroup-nested-object-title-placement

Placement of formal object titles within a formalgroup.

Variable: {<http://docbook.org/ns/docbook/variables>}formalgroup-nested-object-title-placement

Defined in: [modules/variable.xsl](#)

Used in: [modules/blocks.xsl](#), [modules/tablecals.xsl](#)

## Synopsis

`$v:formalgroup-nested-object-title-placement as map(xs:string,xs:string) := fp:parse-key-value-pairs()`

## Description

This variable, usually derived from `$formalgroup-nested-object-title-placement` is a map from element (local) names to placements. The placement is either “before” or “after”.

# \$v:highlight-js-head-elements

CSS and JavaScript to support highlight.js.

Variable: {http://docbook.org/ns/docbook/variables}highlight-js-head-elements

Defined in modules/variable.xsl

Used in: modules/head.xsl

## Synopsis

```
<xsl:variable name="v:highlight-js-head-elements"
    as="element()"">
    <link rel="stylesheet"
        href="${resource-base-uri}css/highlight-11.6.0.min.css"/>
    <script src="${resource-base-uri}js/highlight-11.6.0.min.js"/>
    <script>hljs.highlightAll();</script>
</xsl:variable>
```

## Description

If the “highlight.js” syntax highlighter is selected (see `$verbatim-syntax-highlighter`), these elements are added to the `head` element to load and configure the highlighter.

Starting with the *DocBook xslTNG Stylesheets* version 1.4.1, any `link` elements will be added to the `head` element *before* links to user-defined CSS files so that property overrides may be specified.

# \$v:image-nominal-height

Nominal height of an image.

Variable: {http://docbook.org/ns/docbook/variables}image-nominal-height

Defined in: modules/variable.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport()

## Synopsis

```
$v:image-nominal-height := f:parse-length($image-nominal-height)
```

## Description

If the extension functions necessary to determine the intrinsic height of an image are unavailable, or if the height cannot be determined, this value will be used as the assumed intrinsic height of the image. This is generally the parsed value of \$image-nominal-height .

# \$v:image-nominal-width

Nominal width of an image.

Variable: {http://docbook.org/ns/docbook/variables}image-nominal-width

Defined in: modules/variable.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport()

## Synopsis

```
$v:image-nominal-width := f:parse-length($image-nominal-width)
```

## Description

If the extension functions necessary to determine the intrinsic width of an image are unavailable, or if the width cannot be determined, this value will be used as the assumed intrinsic width of the image. This is generally the parsed value of `$image-nominal-width`.

# \$v:invisible-characters

A list of characters that are invisible in verbatim environments.

Variable: {http://docbook.org/ns/docbook/variables}invisible-characters

Defined in: modules/verbatim.xsl

Used in: modules/verbatim.xsl

## Synopsis

```
$v:invisible-characters := (",",",",",",",",",",",",",",",",",",",")
```

## Description

When callouts are being inserted into a verbatim environment, the stylesheets must count the characters in each line in order to find the correct column. The characters in this list are considered invisible and do not increase the column count as they're passed.

# \$v:localization-base-uri

Base URI for localization data files.

Variable: {http://docbook.org/ns/docbook/variables}localization-base-uri

Defined in: modules/variable.xsl

Used in: modules/functions.xsl

Used by: f:l10n-language()

Since: 2.0.0

## Synopsis

```
$v:localization-base-uri := resolve-uri('../locale/', static-base-uri())
```

## Description

This is the base URI used to resolve references to localization data.

# \$v:media-type-default

Default media type.

Variable: {http://docbook.org/ns/docbook/variables}media-type-default

Defined in: modules/variable.xsl

Used in: modules/objects.xsl

## Synopsis

```
$v:media-type-default as xs:string := 'application/octet-stream'
```

## Description

When attempting to determine the media type of a file, the stylesheets use the `$v:media-type-map` to check for a media type based on the filename (or URI) extension. If there's no extension or if the extension isn't in that map, this value is used.

# \$v:media-type-map

Mapping from extensions to media types.

Variable: {http://docbook.org/ns/docbook/variables}media-type-map

Defined in: modules/variable.xsl

Used in: modules/objects.xsl

# Synopsis

```
<xsl:variable name="v:media-type-map"
    as="map(xs:string, xs:string)">
<xsl:map>
    <xsl:map-entry key=".aac" select="audio/aac"/>
    <xsl:map-entry key=".abw" select="application/x-abiword"/>
    <xsl:map-entry key=".arc" select="application/x-freearc"/>
    <xsl:map-entry key=".avif" select="image/avif"/>
    <xsl:map-entry key=".avi" select="video/x-msvideo"/>
    <xsl:map-entry key=".azw" select="application/vnd.amazon.ebook"/>
    <xsl:map-entry key=".bin" select="application/octet-stream"/>
    <xsl:map-entry key=".bmp" select="image/bmp"/>
    <xsl:map-entry key=".bz" select="application/x-bzip"/>
    <xsl:map-entry key=".bz2" select="application/x-bzip2"/>
    <xsl:map-entry key=".cda" select="application/x-cdf"/>
    <xsl:map-entry key=".csh" select="application/x-csh"/>
    <xsl:map-entry key=".css" select="text/css"/>
    <xsl:map-entry key=".csv" select="text/csv"/>
    <xsl:map-entry key=".doc" select="application/msword"/>
    <xsl:map-entry key=".docx"
        select="application/vnd.openxmlformats-officedocument.wordprocessingml.document"/>
    <xsl:map-entry key=".eot" select="application/vnd.ms-fontobject"/>
    <xsl:map-entry key=".epub" select="application/epub+zip"/>
    <xsl:map-entry key=".gz" select="application/gzip"/>
    <xsl:map-entry key=".gif" select="image/gif"/>
    <xsl:map-entry key=".htm" select="text/html"/>
    <xsl:map-entry key=".html" select="text/html"/>
    <xsl:map-entry key=".ico" select="image/vnd.microsoft.icon"/>
    <xsl:map-entry key=".ics" select="text/calendar"/>
    <xsl:map-entry key=".jar" select="application/java-archive"/>
    <xsl:map-entry key=".jpeg" select="image/jpeg"/>
    <xsl:map-entry key=".jpg" select="image/jpeg"/>
    <xsl:map-entry key=".js" select="text/javascript"/>
    <xsl:map-entry key=".json" select="application/json"/>
    <xsl:map-entry key=".jsonld" select="application/ld+json"/>
    <xsl:map-entry key=".mid" select="audio/midi"/>
```

## Description

When attempting to determine the media type of a file, the stylesheets use this map to check for a media type based on the filename (or URI) extension.

This mapping is derived from Mozilla's list of common types ([https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\\_of\\_HTTP/MIME\\_types/Common\\_types](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types)). The “`.3gp`” and “`.3g2`” extensions have been removed because they have multiple mappings. The “`.text`”, “`.mov`”, “`.m3u8`”, and “`.wmv`” extensions have been added. The media type for “`.svg`” has been simplified to just “`image/svg`” because that seems to be what EPUB readers require.

# \$v:mediaobject-details-placement

Placement of formal object titles.

Variable: {http://docbook.org/ns/docbook/variables}mediaobject-details-placement  
Defined in modules/variable.xsl  
Used in: modules/objects.xsl

## Synopsis

```
$v:mediaobject-details-placement as map(xs:string, xs:string) := fp:parse-key-value-pairs( tokenize($mediaobject-details-placement, " ") )
```

## Description

This variable, usually derived from `$mediaobject-details-placement` is a map from element (local) names to placements. The placement is either “before” or “after”.

# \$v:mediaobject-exclude-extensions

Excluded media type extensions.

Variable: {http://docbook.org/ns/docbook/variables}mediaobject-exclude-extensions

Defined in modules/variable.xsl

Used in: modules/objects.xsl

## Synopsis

```
$v:mediaobject-exclude-extensions := tokenize($mediaobject-exclude-extensions, '\s+')
```

## Description

This variable simply contains the list of excluded extensions constructed from the `$mediaobject-exclude-extensions` parameter.

# \$v:mediaobject-input-base-uri

Base URI of images and other media in the XML sources.

## Description

This variable was replaced with `f:mediaobject-input-base-uri()`.

# \$v:mediaobject-output-base-uri

Base URI of images and other media in the output.

Variable: {http://docbook.org/ns/docbook/variables}mediaobject-output-base-uri

Defined in modules/variable.xsl

Used in: Not used.

## Synopsis

```
<xsl:variable name="v:mediaobject-output-base-uri"
               as="xs:string?">
  <xsl:message use-when=""mediaobject-uris' = $v:debug"
               select=""Mediaobject out. base URI:",           if (empty($mediaobject-output-base-uri))
  <xsl:sequence select="if (empty($mediaobject-output-base-uri))           then ()
</xsl:variable>
```

## Description

This value is calculated from the `$mediaobject-output-base-uri` parameter. If the parameter is the empty string, then this value is the empty sequence. Otherwise, this value is the value of the `$mediaobject-output-base-uri` parameter, which is assumed to be absolute. A trailing slash will be added to the parameter value if it is not present.

# \$v:nominal-page-width

The nominal page width.

Variable: {http://docbook.org/ns/docbook/variables}nominal-page-width

Defined in: modules/variable.xsl

Used in: modules/tablecals.xsl

## Synopsis

```
$v:nominal-page-width := f:parse-length($nominal-page-width)
```

## Description

To calculate the width of the columns in some complex CALS tables, the stylesheets need to know the page width. The `$nominal-page-width` is used for this value. It is generally the value of `$nominal-page-width` parsed as a length (see *Section 5.2, “Lengths and units”*).

# \$v:olink-databases

External olink databases.

Variable:	{http://docbook.org/ns/docbook/variables}olink-databases
Defined in:	modules/variable.xsl
Used in:	modules/links.xsl

## Synopsis

```
<xsl:variable name="v:olink-databases"
    as="element(h:targetdb)*">
    <xsl:if test="normalize-space($olink-databases) != "">
        <xsl:for-each select="tokenize($olink-databases, ',\s*') ! normalize-space(.)">
            <xsl:variable name="db" select="."/>
            <xsl:try>
                <xsl:variable name="olinkdb" select="doc($db)/h:targetdb"/>
                <xsl:if test="empty($olinkdb)">
                    <xsl:message select="No targets in olinkdb:', $db"/>
                </xsl:if>
                <xsl:sequence select="$olinkdb"/>
                <xsl:catch>
                    <xsl:message select="Failed to load olinkdb:', $db"/>
                </xsl:catch>
            </xsl:try>
        </xsl:for-each>
    </xsl:if>
</xsl:variable>
```

## Description

The `$v:olink-databases` variable is usually derived from the `$olink-databases` parameter. It must contain a sequence of olink target databases.

# \$v:prism-js-head-elements

CSS and JavaScript to support Prism.

Variable: {http://docbook.org/ns/docbook/variables}prism-js-head-elements

Defined in modules/variable.xsl

Used in: modules/head.xsl

## Synopsis

```
<xsl:variable name="v:prism-js-head-elements"
    as="element()**">
    <link rel="stylesheet" href="{$resource-base-uri}css/prism.css"/>
    <script src="{$resource-base-uri}js/prism.js"/>
</xsl:variable>
```

## Description

If the “prism” syntax highlighter is selected (see `$verbatim-syntax-highlighter`), these elements are added to the `head` element to load and configure the highlighter.

Starting with the *DocBook xslTNG Stylesheets* version 1.4.1, any `link` elements will be added to the `head` element *before* links to user-defined CSS files so that property overrides may be specified.

# \$v:standard-transforms

The standard pre-processing transformations.

Variable: {http://docbook.org/ns/docbook/variables}standard-transforms

Defined in: docbook.xsl

Used in: docbook.xsl

# Synopsis

```
<xsl:variable name="v:standard-transforms"
    as="map(*)">
<xsl:map>
    <xsl:map-entry key="stylesheet-location"
        select="resolve-uri('transforms/00-logstruct.xsl', static-base-uri())"/>
</xsl:map>
<xsl:map>
    <xsl:map-entry key="stylesheet-location"
        select="resolve-uri('transforms/10-xinclude.xsl', static-base-uri())"/>
    <xsl:map-entry key="functions"
        select="Q{http://docbook.org/extensions/xslt}xinclude"/>
    <xsl:map-entry key="test" select="exists(/xi:include)"/>
</xsl:map>
<xsl:map>
    <xsl:map-entry key="stylesheet-location"
        select="resolve-uri('transforms/20-db4to5.xsl', static-base-uri())"/>
    <xsl:map-entry key="test">
        not(namespace-uri(/*) = 'http://docbook.org/ns/docbook')
    </xsl:map-entry>
    <xsl:map-entry key="extra-params"
        select="map { QName(., 'base-uri'): 'base-uri(/*)' }"/>
</xsl:map>
<xsl:map>
    <xsl:map-entry key="stylesheet-location"
        select="resolve-uri('transforms/30-transclude.xsl', static-base-uri())"/>
    <xsl:map-entry key="test" select="f:is-true($docbook-transclusion)"/>
</xsl:map>
<xsl:map>
    <xsl:map-entry key="stylesheet-location"
        select="resolve-uri('transforms/40-profile.xsl', static-base-uri())"/>
    <xsl:map-entry key="test">
        f:is-true($dynamic-profiles)
        or $profile-lang != "          or $profile-revisionflag != "
        or $profile-role != "      or $profile-arch != "
        or $profile-audience != "    or $profile-condition != "
    </xsl:map-entry>
</xsl:map>
```

## Description

This variable contains the list of preprocessing transforms applied to each document. You can add transformations that operate on the original document ( `$transform-original` ) or on the result of these transforms, but before the DocBook HTML transformation occurs ( `$transform-before` ), or after the HTML transformation ( `$transform-after` ).

You shouldn't need to modify this variable unless you want to remove or reorder the standard transforms, or insert your own into the middle.

# \$v:templates

Custom templates for title pages.

Variable: {http://docbook.org/ns/docbook/variables}templates

Defined in: modules/templates.xsl

Used in: modules/templates.xsl

## Synopsis

```
<xsl:variable name="v:templates"
    as="document-node()">
    <xsl:document/>
</xsl:variable>
```

## Description

The stylesheets use templates, as described in *Section 5.5, “Templates”*, to determine the formatting of title pages<sup>1</sup>. Any templates provided in `$v:templates` will be used preferentially to whatever builtin templates exist.

---

<sup>1</sup> The term “title pages” should be understood very broadly here. Anything that has a title has a title page, even if that “page” consists of only a single heading.

# \$v:theme-list

Enumerates available themes.

Variable:	{http://docbook.org/ns/docbook/variables}theme-list
Defined in:	modules/variable.xsl
Used in:	modules/variable.xsl

## Synopsis

```
<xsl:variable name="v:theme-list"
    as="element()*"
    <theme name="Materials dark" id="materials-dark" dark="true"/>
    <theme name="Materials light" id="materials-light" dark="false"/>
</xsl:variable>
```

## Description

This variable is part of the experimental themes feature. It enumerates the available themes. Each theme has three parts, a name, an ID, and (optionally) an indication of whether or not it's a "dark" theme.

Themes are implemented in CSS. If the `$theme-picker` is enabled, it will be possible for the reader to select a theme. The theme names are used for this purpose.

The ID value is used as a class on the `html` element. Everything else about the theme is implemented in CSS. To add a new theme, provide a set of CSS rules that apply when the ID class is in effect.

The dark mode setting is only used as a default. If the document does not specify a `$default-theme`, if the user has never selected a theme, if the user has enabled dark mode at the operating system level, and if the browser makes this fact available, the first "dark" theme will automatically be selected.

# \$v:title-groups

Controls how titles are formatted.

Variable: {http://docbook.org/ns/docbook/variables}title-groups

Defined in: modules/titles.xsl

Used in: modules/titles.xsl

Since: 2.0.0

# Synopsis

```
<xsl:variable name="v:title-groups"
    as="element()+
<xsl:sequence select="$v:user-title-groups"/>
<title xpath="self::db:section|self::db:sect1
    group="{if (f:is-true($section-numbers)) then 'title-numbered'
    else 'title-unnumbered'}" />
<title xpath="self::db:article|self::db:preface|self::db:chapter|self::db:appendix"
    group="{if (f:is-true($component-numbers)) then 'title-numbered'
    else 'title-unnumbered'}" />
<title xpath="self::db:set" group="title-unnumbered"/>
<title xpath="self::db:book|self::db:part|self::db:reference"
    group="{if (f:is-true($division-numbers)) then 'title-numbered'
    else 'title-unnumbered'}" />
<title xpath="self::db:figure[parent::db:formalgroup]
    group="subfigure-title"/>
<title xpath="self::db:figure|self::db:table|self::db:equation|self::db:example"
    group="title-numbered"/>
<title xpath="self::db:formalgroup" group="title-numbered"/>
<title xpath="self::db:step|self::db:listitem[parent::db:orderedlist]"
    group="title-unnumbered"/>
<title xpath="self::db:glosssee|self::db:glossseealso"
    group="title-unnumbered"/>
<title xpath="self::db:see|self::db:seealso" group="title-unnumbered"/>
<title xpath="self::db:question|self::db:answer" group="title-numbered"/>
<title xpath="self::*" group="title-unnumbered"/>
</xsl:variable>
```

# Description

The title groups variable determines which template group is used for a title. See *Chapter 4, Localization*.

# ~~\$v:title-properties~~

Controls how titles are formatted.

## Description

Replaced by `$v:title-groups`.

# \$v:titlepage-default

Default template for title pages.

Variable: {http://docbook.org/ns/docbook/variables}titlepage-default

Defined in: modules/templates.xsl

Used in: modules/templates.xsl

## Synopsis

```
<xsl:variable name="v:titlepage-default"
               as="element()">
  <titlepage-default>
    <header>
      <apply-templates select="db:title">
        <div class="title">
          <content/>
        </div>
      </apply-templates>
    </header>
  </titlepage-default>
</xsl:variable>
```

## Description

Title pages, taken loosely to mean anything with a `title`, are formatted using templates, see *Section 5.5, “Templates”*. If a template doesn’t exist for a particular context, the `$v:titlepage-default` template is used.

# \$v:toc-close

Persistent ToC close button.

Variable:	{http://docbook.org/ns/docbook/variables}toc-close
Defined in:	modules/variable.xsl
Used in:	modules/chunk-output.xsl

## Synopsis

```
<xsl:variable name="v:toc-close"
    as="element()>
    
</xsl:variable>
```

## Description

The contents of this variable will be rendered as the “close” icon for the persistent ToC.

# \$v:toc-open

Persistent ToC open button.

Variable: {http://docbook.org/ns/docbook/variables}toc-open

Defined in: modules/variable.xsl

Used in: modules/chunk-output.xsl

## Synopsis

```
<xsl:variable name="v:toc-open"
    as="element()">
    
</xsl:variable>
```

## Description

The contents of this variable will be rendered as the “open” icon for the persistent ToC.

# \$v:unit-scale

List of known measurement units and their sizes.

Variable: {http://docbook.org/ns/docbook/variables}unit-scale

Defined in: modules/units.xsl

Used in: modules/units.xsl

Used by: f:parse-length(), f:absolute-length()

## Synopsis

```
<xsl:variable name="v:unit-scale"
    as="map(*)">
    <xsl:map>
        <xsl:map-entry key=""px"" select="1.0"/>
        <xsl:map-entry key=""in"" select="$pixels-per-inch"/>
        <xsl:map-entry key=""m"" select="$pixels-per-inch div 2.54 * 100.0"/>
        <xsl:map-entry key=""cm"" select="$pixels-per-inch div 2.54"/>
        <xsl:map-entry key=""mm"" select="$pixels-per-inch div 25.4"/>
        <xsl:map-entry key=""pt"" select="$pixels-per-inch div 72.0"/>
        <xsl:map-entry key=""pc"" select="$pixels-per-inch div 6.0"/>
        <xsl:map-entry key=""em"" select="$pixels-per-inch div 6.0"/>
        <xsl:map-entry key=""barleycorn"" select="$pixels-per-inch div 3.0"/>
    </xsl:map>
</xsl:variable>
```

## Description

The `$v:unit-scale` map contains a set of measurement units and their absolute length in terms of pixels (as computed with `$pixels-per-inch`).

Recognized units at the time of this writing are shown in *Figure 1, “Recognized units of measure”*.

Unit	Size
px	1.0
in	\$pixels-per-inch
m	\$pixels-per-inch div 2.54 * 100.0
cm	\$pixels-per-inch div 2.54
mm	\$pixels-per-inch div 25.4
pt	\$pixels-per-inch div 72.0
pc	\$pixels-per-inch div 6.0
em	\$pixels-per-inch div 6.0
barleycorn	\$pixels-per-inch div 3.0

**Figure 1. Recognized units of measure**

See also *Section 5.2, “Lengths and units”*.

# \$v:user-title-groups

Controls how titles are formatted.

Variable: {http://docbook.org/ns/docbook/variables}user-title-groups

Defined in: modules/titles.xsl

Used in: modules/titles.xsl

Used by: \$v:title-groups

Since: 2.0.0

## Synopsis

```
<xsl:variable name="v:user-title-groups"  
              as="element()"/>
```

## Description

This variable contains user-supplied overrides to \$v:title-groups . See that variable for more details.

# ~~\$v:user-title-properties~~

Controls how titles are formatted.

## Description

Replaced by `$v:user-title-groups`.

# \$v:user-xref-groups

Controls how cross-references are formatted.

Variable: {http://docbook.org/ns/docbook/variables}user-xref-groups

Defined in: modules/xref.xsl

Used in: modules/xref.xsl

Used by: \$v:xref-groups

Since: 2.0.0

## Synopsis

```
<xsl:variable name="v:user-xref-groups"
               as="element()"/>
```

## Description

This variable contains user-supplied overrides to `$v:xref-groups`. See that variable for more details.

# ~~\$v:user-xref-properties~~

Controls how cross-references are formatted.

## Description

Replaced by `$v:user-xref-groups`.

# \$v:verbatim-callouts

A list determining how callouts are processed.

Variable:	{http://docbook.org/ns/docbook/variables}verbatim-callouts
Defined in:	modules/variable.xsl
Used in:	modules/verbatim.xsl
Used by:	\$v:verbatim-properties

## Synopsis

```
$v:verbatim-callouts as xs:string* := tokenize($verbatim-callouts, '\s+')
```

## Description

This value is a sequence constructed automatically from the `$verbatim-callouts` parameter.

# \$v:verbatim-line-style

List of verbatim elements to be rendered in the line style.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-line-style

Defined in: modules/variable.xsl

Used in: modules/verbatim.xsl

Used by: \$v:verbatim-properties

## Synopsis

```
$v:verbatim-line-style := tokenize($verbatim-line-style, '\s+')
```

## Description

The elements whose local names appear in this list will be formatted using the line verbatim style by default. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

This list is usually constructed from the `$verbatim-line-style` parameter.

# \$v:verbatim-number-every-nth

Line numbering frequency.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-number-every-nth

Defined in modules/variable.xsl

Used in: modules/verbatim.xsl

## Synopsis

```
$v:verbatim-number-every-nth := xs:integer($verbatim-number-every-nth)
```

## Description

When formatting verbatim environments with line numbers, every `$verbatim-number-every-nth` line is numbered.

This value is usually constructed from the `$verbatim-number-every-nth` parameter.

# \$v:verbatim-number-first-line

Always number the first line of a verbatim listing?

Variable: {http://docbook.org/ns/docbook/variables}verbatim-number-first-line

Defined in modules/variable.xsl

Used in: modules/verbatim.xsl

## Synopsis

`$v:verbatim-number-first-line := f:is-true($verbatim-number-first-line)`

## Description

When formatting verbatim environments with line numbers, the first line will be numbered (irrespective of the setting of `$v:verbatim-number-every-nth` if this value is true).

This value is usually constructed from the `$verbatim-number-first-line` parameter.

# \$v:verbatim-number-minlines

Shortest listing to number.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-number-minlines

Defined in modules/variable.xsl

Used in: modules/verbatim.xsl

## Synopsis

`$v:verbatim-number-minlines := xs:integer($verbatim-number-minlines)`

## Description

When formatting verbatim environments with line numbers, environments less than `$verbatim-number-minlines` in length will not be numbered.

This value is usually constructed from the `$verbatim-number-minlines` parameter.

# \$v:verbatim-numbered-elements

Identifies verbatim elements that should have line numbers.

Variable:	{http://docbook.org/ns/docbook/variables}verbatim-numbered-elements
Defined in	modules/variable.xsl
Used in:	modules/verbatim.xsl
Used by:	\$v:verbatim-properties

## Synopsis

`$v:verbatim-numbered-elements as xs:string* := tokenize($verbatim-numbered-elements, '\s+')`

## Description

A list of the verbatim environments that should be numbered.

This value is usually constructed from the `$verbatim-numbered-elements` parameter.

# \$v:verbatim-plain-style

List of verbatim elements to be rendered in the plain style.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-plain-style

Defined in: modules/variable.xsl

Used in: modules/verbatim.xsl

Used by: \$v:verbatim-properties

## Synopsis

```
$v:verbatim-plain-style as xs:string* := tokenize($verbatim-plain-style, '\s+')
```

## Description

The elements whose local names appear in this list will be formatted using the plain verbatim style by default. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

This list is usually constructed from the `$verbatim-plain-style` parameter.

# \$v:verbatim-properties

Controls the verbatim properties of an element.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-properties

Defined in: modules/verbatim.xsl

Used in: modules/verbatim.xsl

## Synopsis

`$v:verbatim-properties as array(map(*))`

## Description

When processing a verbatim environment, this array of maps is consulted to determine what properties apply to the verbatim environment. Most simple, global customizations of verbatim environments can be achieved by setting the `$verbatim-line-style`, `$verbatim-plain-style`, `$verbatim-callouts`, and `$verbatim-numbered-elements` parameters.

# \$v:verbatim-space

The space character to use when padding verbatim lines.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-space

Defined in: modules/variable.xsl

Used in: modules/verbatim.xsl

## Synopsis

```
<xsl:variable name="v:verbatim-space"
    as="node()">
    <xsl:value-of select="substring($verbatim-space || ' ', 1, 1)" />
</xsl:variable>
```

## Description

When a verbatim line has to be padded in order to get a callout to appear in the correct column, this character will be used as the padding character. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

This character is usually constructed from the `$verbatim-space` parameter. Unlike the `$verbatim-space`, this variable must be a node not a string.

# \$v:verbatim-syntax-highlight-languages

Languages for which syntax highlighting should be performed.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-syntax-highlight-languages

Defined in: modules/variable.xsl

Used in: modules/head.xsl, modules/verbatim.xsl

Used by: f:highlight-verbatim()

## Synopsis

```
$v:verbatim-syntax-highlight-languages := tokenize($verbatim-syntax-highlight-languages, '\s+')
```

## Description

The `$verbatim-syntax-highlight-languages` is a single string for the convenience of users. This variable contains the languages as a list.

# \$v:verbatim-syntax-highlight-options

Syntax highlighting options.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-syntax-highlight-options

Defined in modules/variable.xsl

Used in: modules/verbatim.xsl

## Synopsis

```
$v:verbatim-syntax-highlight-options := map { }
```

## Description

If the Pygments (<https://pygments.org/>) syntax highlighter is applied, the `$verbatim-syntax-highlight-options` are applied.

Three options are recognized:

`language`

The programming language to use for syntax highlighting. This option is required.

`show-command`

If the `show-command` option is true (“1”, “yes”, or “true”), then the `pygmentize` command will be printed. This is a debugging option.

`show-result`

If the `show-result` option is true (“1”, “yes”, or “true”), then the string returned from `pygmentize` will be printed. This is a debugging option.

**Note**

Syntax highlighting with Pygments requires spawning an external process for each highlighted environment. This can add significantly to the processing time if there are many such environments.

# \$v:verbatim-syntax-highlight-pygments-options

Pygments options.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-syntax-highlight-pygments-options

Defined in: modules/variable.xsl

Used in: modules/verbatim.xsl

## Synopsis

```
$v:verbatim-syntax-highlight-pygments-options := map { }
```

## Description

If the Pygments (<https://pygments.org/>) syntax highlighter is applied, the `$verbatim-syntax-highlight-pygments-options` are applied.

Each key/value pair in the map will be passed to `pygmentize` as a `-Pkey=value` option.

# \$v:verbatim-table-style

List of verbatim elements to be rendered in the table style.

Variable: {http://docbook.org/ns/docbook/variables}verbatim-table-style

Defined in: modules/variable.xsl

Used in: modules/verbatim.xsl

Used by: \$v:verbatim-properties

## Synopsis

```
$v:verbatim-table-style := tokenize($verbatim-table-style, '\s+')
```

## Description

The elements whose local names appear in this list will be formatted using the line verbatim style by default. For a discussion of verbatim elements and styles, see *Section 5.3, “Verbatim styles”*.

This list is usually constructed from the `$verbatim-table-style` parameter.

# \$v:xref-groups

Controls how cross-references are formatted.

Variable:	{http://docbook.org/ns/docbook/variables}xref-groups
Defined in:	modules/xref.xsl
Used in:	modules/xref.xsl
Since:	2.0.0

## Synopsis

```
<xsl:variable name="v:xref-groups"
    as="element()"+>
    <xsl:sequence select="$v:user-xref-groups"/>
    <crossref xpath="self::db:section[ancestor::db:preface]"
        group="xref"
        template="section-in-preface"/>
    <crossref xpath="self::db:section" group="xref-number-and-title"/>
    <crossref xpath="self::db:chapter|self::db:appendix"
        group="xref-number-and-title"/>
    <crossref xpath="self::db:part|self::db:reference" group="xref-number-and-title"/>
    <crossref xpath="self::db:figure|self::db:example|self::db:table
        |self::db:procedure|sel
        group="xref-number-and-title"/>
    <crossref xpath="self::*" group="xref"/>
</xsl:variable>
```

## Description

The cross-reference groups variable determines which template group is used for a cross-reference. See *Chapter 4, Localization*.

# ~~\$v:xref-properties~~

Controls how cross-references are formatted.

## Description

Replaced by `$v:xref-groups`.



# III. Function reference

# ext:cwd

Returns the current working directory.

Function: {<http://docbook.org/extensions/xslt>}cwd#0

## Synopsis

`ext:cwd()` as `xs:string`

## Description

This extension function returns the current working directory from which the processor was executed.

# ext:image-metadata

Return intrinsic properties of an image.

Function: {<http://docbook.org/extensions/xslt>}image-metadata#1

## Synopsis

`ext:image-metadata($filename as xs:string) as map(*)`

## Description

This extension function returns a map of metadata properties about the image, including its intrinsic size: its width and height in pixels. If the size can be determined, the map returned will have a “width” key whose value is the width of the image and a “height” key whose value is the height of the image. Both sizes will be integer values.

This function uses the *metadata-extractor* library. The map may contain other properties as well.

- If the media was successfully parsed with *metadata-extractor*, all of the properties identified by the extractor will be returned. Each property name will be converted to lowercase and spaces will be replaced with hyphens. (In other words, a property with the tag name “Palette Colour Count” will appear in the map with a key of “palette-colour-count”.)

Property values that appear to be data/time values<sup>1</sup> will be converted to strings that are valid ISO 8601 date/time values. Any control characters that can't appear in XML but that appear in a value will be replaced with Unicode references, \uxxxx. For example, a null byte will be replaced with \u0000.

- If the media isn't successfully parsed with *metadata-extractor*,
  - If it appears to be a PDF document, the function will search for the “MediaBox” or “CropBox” in the first 100 lines of the PDF. If either is found, its dimensions are used to calculate the width and height. If both are present, CropBox is preferred.

<sup>1</sup> A value that matches the regular expression “^\d\d\d\d:\d\d:\d\d\d\d\$” is assumed to be a date/time. This is the format of date/time values used in EXIF metadata for most properties that have a date/time value.

- If it appears to be an SVG document, the function will search for `width` and `height` attributes on the root element. All of the attributes of the root element will be returned as properties, using Clark names.

If the `width` and `height` properties exist, they will always be integers. This means that if an image has a property with the name `width` or `height` and its value isn't an integer, it will not be returned.

If the `$debug` parameter includes the token `image-properties`, the stylesheets will print every property returned in an `xsl:message`.

# ext:image-properties

Return intrinsic properties of an image.

Function: {<http://docbook.org/extensions/xslt>}image-properties#1

## Synopsis

`ext:image-properties( $filename as xs:string) as map(*)`

## Description

This extension function returns the intrinsic size of an image: its width and height in pixels. If the size can be determined, the map returned will have a “width” key whose value is the width of the image and a “height” key whose value is the height of the image. Both sizes will be integral numbers of pixels.

The `ext:image-metadata()` function provides better results. It should be preferred unless the *metadata-extractor* libraries are unavailable.

# ext:pygmentize

Syntax highlight a listing with pygmentize.

Function:	{http://docbook.org/extensions/xslt}pygmentize#1
	{http://docbook.org/extensions/xslt}pygmentize#2
	{http://docbook.org/extensions/xslt}pygmentize#3

## Synopsis

`ext:pygmentize( $listing as xs:string) as xs:string`

`ext:pygmentize( $listing as xs:string,  
$options as item()) as xs:string`

`ext:pygmentize( $listing as xs:string,  
$options as item(),  
$pygments-options as map(xs:string,xs:string)) as xs:string`

## Description

This extension function runs the `pygmentize` command to add syntax highlighting to a listing.

The second argument can either be map, in which case the key-value pairs of the map constitute the options, or it can be a single string. If it's a single string, it's interpreted as if it was a map with the key “`language`” and the string as the value.

The third argument must be a map. Each key-value pair will be passed to `pygmentize` as “`-P key = value`”.

The function returns the string output from `pygmentize`. It must be parsed with `parse-xml` if you wish to process it as XML.

Be aware that `pygmentize` strips leading blank lines off of the source listing. If you're selecting portions of a listing with XInclude, make sure not to select leading blank lines in the listing if you are trying to accurately count the lines.

# ext:pygmentize-available

Returns true if pygmentize is available.

Function: {http://docbook.org/extensions/xslt}pygmentize-available#0

## Synopsis

`ext:pygmentize-available()` as `xs:boolean`

## Description

In order to run Pygments on listings, the `ext:pygmentize()` extension function must be available *and* the `pygmentize` command must be available on the host system.

This function returns true if it successfully finds the `pygmentize` command on the system path.

# ext:validate-with-relax-ng

Returns the current working directory.

```
Function: {http://docbook.org/extensions/xslt}validate-with-relax-  
ng#2  
{http://docbook.org/extensions/xslt}validate-with-relax-  
ng#3
```

## Synopsis

```
ext:validate-with-relax-  
ng( $node as node(),  
    $schema as item()) as map(xs:string, item())
```

```
ext:validate-with-relax-  
ng(  
    $node as node(),  
    $schema as item(),  
    $options as map(xs:string, xs:string)) as map(xs:string,  
        item())
```

## Description

This extension function validates the `$node` provided against the `$schema`. The `$schema` can be either a string (the URI of the RELAX NG grammar file) or a node (a RELAX NG grammar document). There is no support for the RELAX NG Compact Syntax at this time.

The available options are `assert-valid` which defaults to `true()` and `dtd-compatibility` which isn't actually supported yet.

If `assert-valid` is `true()` and the document is not valid according to the grammar provided, an exception is raised.

If `assert-valid` is `false()`, or the document is valid, the map returned will contain the following keys:

`valid`

A boolean indicating whether or not the document was valid.

`document`

The validated document. Today, this always returns the same node, but it may eventually return an augmented document.

## errors

An array of maps containing the `type` (`warning`, `error`, or `fatal-error`), `message`, `line`, and `column` where an error occurred. If the document is valid, the `errors` key will not be present.

# ext:xinclude

Performs XInclude processing.

Function:	{http://docbook.org/extensions/xslt}xinclude#1
	{http://docbook.org/extensions/xslt}xinclude#2

## Synopsis

**ext:xinclude(** \$node **as node()) as** node()\*

**ext:xinclude(** \$node **as node(),**  
\$options **as map(xs:QName, item())\*) as** node()\*

## Description

Performs XInclude processing on node and all of its descendants. The options “fixup-xml-base” and “fixup-xml-lang” are true() by default.

# f:absolute-length

Returns the absolute length of a unit of measurement.

Function: {<http://docbook.org/ns/docbook/functions>}absolute-length#1

Defined in: modules/units.xsl

Used in: modules/objects.xsl, modules/tablecals.xsl

Used by: f:css-length()

## Synopsis

**f:absolute-length( \$length ) as map(\*) as xs:double**

## Description

For a given unit of measurement, returns the absolute length in terms of pixels. For a discussion of units, see *Section 5.2, “Lengths and units”*.

See also `$pixels-per-inch`, `$default-length-magnitude`, and `$default-length-unit`.

# f:attributes

Returns the attribute that apply to an output element.

Function: {<http://docbook.org/ns/docbook/functions>}attributes#2

{<http://docbook.org/ns/docbook/functions>}attributes#4

Defined in modules/functions.xsl (2)

Used in: modules/functions.xsl, modules/inlines.xsl, modules/attributes.xsl

Used by: f:attributes(), t:inline

## Synopsis

```
f:attributes( $node as element(),
               $attributes as attribute()*) as attribute()*
```

```
f:attributes( $node as element(),
               $attributes as attribute()*, 
               $extra-classes as xs:string*, 
               $exclude-classes as xs:string*) as attribute()*
```

## Description

Most output elements have attributes: any `xml:id` attributes in the source are reflected in the output as `id` attributes, most elements get a `class` attribute, etc. The `f:attributes()` function is called to determine what those attributes should be.

In practice, the way this usually works is the element is processed in the `m:attributes` mode and that template calls `f:attributes()` to generate the actual attribute nodes.

# f:cals-colsep

Returns the “colsep” value associated with a CALS table cell.

Function: {http://docbook.org/ns/docbook/functions}cals-colsep#3

Defined in: modules/tablecals.xsl

Used in: modules/tablecals.xsl

## Synopsis

```
f:cals-colsep( $row as element(db:row),  
                 $cell as map(*),  
                 $last-col-colsep as xs:boolean) as xs:string?
```

## Description

The column separator, or “colsep”, associated with a CALS table cell depends on a variety of factors: `colspec`, and `spanspec` elements as well as the attributes on the cell itself and its ancestors.

This function returns the value that applies to the specified cell.

# f:cals-rowsep

Returns the “rowsep” value associated with a CALS table cell.

Function: {http://docbook.org/ns/docbook/functions}cals-rowsep#3

Defined in: modules/tablecals.xsl

Used in: modules/tablecals.xsl

## Synopsis

```
f:cals-rowsep( $row as element(db:row),
                 $cell as map(*),
                 $last-row-rowsep as xs:boolean) as xs:string?
```

## Description

The row separator, or “rowsep”, associated with a CALS table cell depends on a variety of factors: `colspec`, and `spanspec` elements as well as the attributes on the cell itself and its ancestors.

This function returns the value that applies to the specified cell.

## f:checkgentext

Returns generated text.

## Description

Generated text has been reworked, see *Chapter 4, Localization*.

# f:chunk

Returns chunking attributes.

Function: {http://docbook.org/ns/docbook/functions}chunk#1

Defined in: modules/chunk.xsl

Used in: modules/attributes.xsl

## Synopsis

`f:chunk( $node as element() ) as attribute()*`

## Description

This function returns the `db-chunk` and perhaps other chunking-related attributes for the given node.

# f:chunk-filename

Returns the filename to use for a particular chunk.

Function: {http://docbook.org/ns/docbook/functions}chunk-filename#1

Defined in: modules/chunk.xsl

Used in: modules/chunk.xsl

Used by: f:chunk()

## Synopsis

`f:chunk-filename( $node ) as element() as xs:string`

## Description

When chunking, see *Section 2.6, ““Chunked” output”*, this function is called to determine the filename for a chunk. The default implementation considers relevant `db` processing instructions (`filename`, `href`, and `basename` pseudo-attributes) as well as the generated ID of the element.

# f:chunk-title

Returns the title of a chunk.

Function:	{http://docbook.org/ns/docbook/functions}chunk-title#1
Defined in:	modules/chunk-cleanup.xsl
Used in:	modules/chunk-cleanup.xsl
Used by:	t:bottom-nav

## Synopsis

`f:chunk-title( $chunk as element()?) as node()*`

## Description

When the title of a neighboring chunk is required (for header or footer navigation, for example), this function can be used to obtain the title of that chunk.

This function must return the HTML-formatted title, not the DocBook title.

# f:css-length

Returns a length in the format of a CSS property.

Function: {http://docbook.org/ns/docbook/functions}css-length#2

Defined in: modules/objects.xsl

Used in: modules/objects.xsl

## Synopsis

```
f:css-length( $property as xs:string,  
               $length as map(*)?) as xs:string?
```

## Description

Returns the CSS property specified with the corresponding length, if the length exists. Returns the empty sequence otherwise.

# f:css-property

Returns a CSS property for a given value.

Function:	{http://docbook.org/ns/docbook/functions}css-property#2
Defined in:	modules/objects.xsl
Used in:	modules/objects.xsl
Used by:	f:css-length()

## Synopsis

```
f:css-property( $property as xs:string,  
                 $value as xs:string?) as xs:string?
```

## Description

If the specified value exists, a string formatted as a CSS property is returned.  
If the value doesn't exist, an empty sequence is returned.

# f:date-format

Returns the format string for a date.

Function: {http://docbook.org/ns/docbook/functions}date-format#1

Defined in: modules/functions.xsl

Used in: modules/inlines.xsl

## Synopsis

`f:date-format( $context as element() ) as xs:string`

## Description

Dates may be stored in a variety of formats and the `(pubdate)` element even allows inline markup. The `f:date-format()` function determines how a date will be formatted:

- If the date contains embedded markup, the special value “`apply-templates`” is returned to indicate that string formatting isn’t appropriate.
- If the date conforms to an *ISO 8601* date, the `$date-date-format` string is returned.
- If the date conforms to an *ISO 8601* dateTime, the `$date-dateTime-format` string is returned.
- If the date does not conform to either of those date formats, “`apply-templates`” is returned.

If the date conforms to a date or dateTime, the author may override the format string by providing a `db` processing instruction with a `date-format` pseudo-attribute.

*Example 1, “Several dates in ISO 8601 formats”* shows an example of several dates that may be formatted in more familiar forms.

```
<para>The Unix epoch begins at  
<date>1970-01-01T00:00:00Z</date>.  
Grace Hopper was born on <date>1906-12-09</date>.  
That was a <date><?db date-format="[F]"?>1906-12-09</date>.  
I was born on a <date>Friday</date>.</para>
```

### Example 1. Several dates in ISO 8601 formats

With default formats, these are formatted as shown:

The Unix epoch begins at 00:00 01 Jan 1970. Grace Hopper was born on 09 Dec 1906. That was a Sunday. I was born on a Friday.

# f:empty-length

Returns a map that represents an empty length.

Function:{http://docbook.org/ns/docbook/functions}empty-length#0

Defined in modules/units.xsl

Used in: modules/units.xsl, modules/objects.xsl

Used by: f:parse-length(), f:object-width(), f:object-height(), f:object-contentwidth(),  
f:object-contentheight()

## Synopsis

`f:empty-length()` as `map(*)`

## Description

Lengths are compound objects. Lengths consist of a magnitude (a number) and a unit: 3 inches, for example, or 11.9 barleycorns<sup>1</sup> in the case of absolute lengths. For relative lengths, the unit is something relative like percent.

These are represented as maps internally and `f:empty-length()` returns a length with no magnitude or units.

---

<sup>1</sup> Yes, that's a real unit. It's equal to  $\frac{1}{3}$  of an inch. I'll leave it to Wikipedia ([https://en.wikipedia.org/wiki/Barleycorn\\_\(unit\)](https://en.wikipedia.org/wiki/Barleycorn_(unit))) to explain the details. Just looking at the chart of imperial units makes me want to add them all!

# f:equal-lengths

Returns true if two lengths are equal.

Function: {<http://docbook.org/ns/docbook/functions>}equal-lengths#2

Defined in: modules/units.xsl

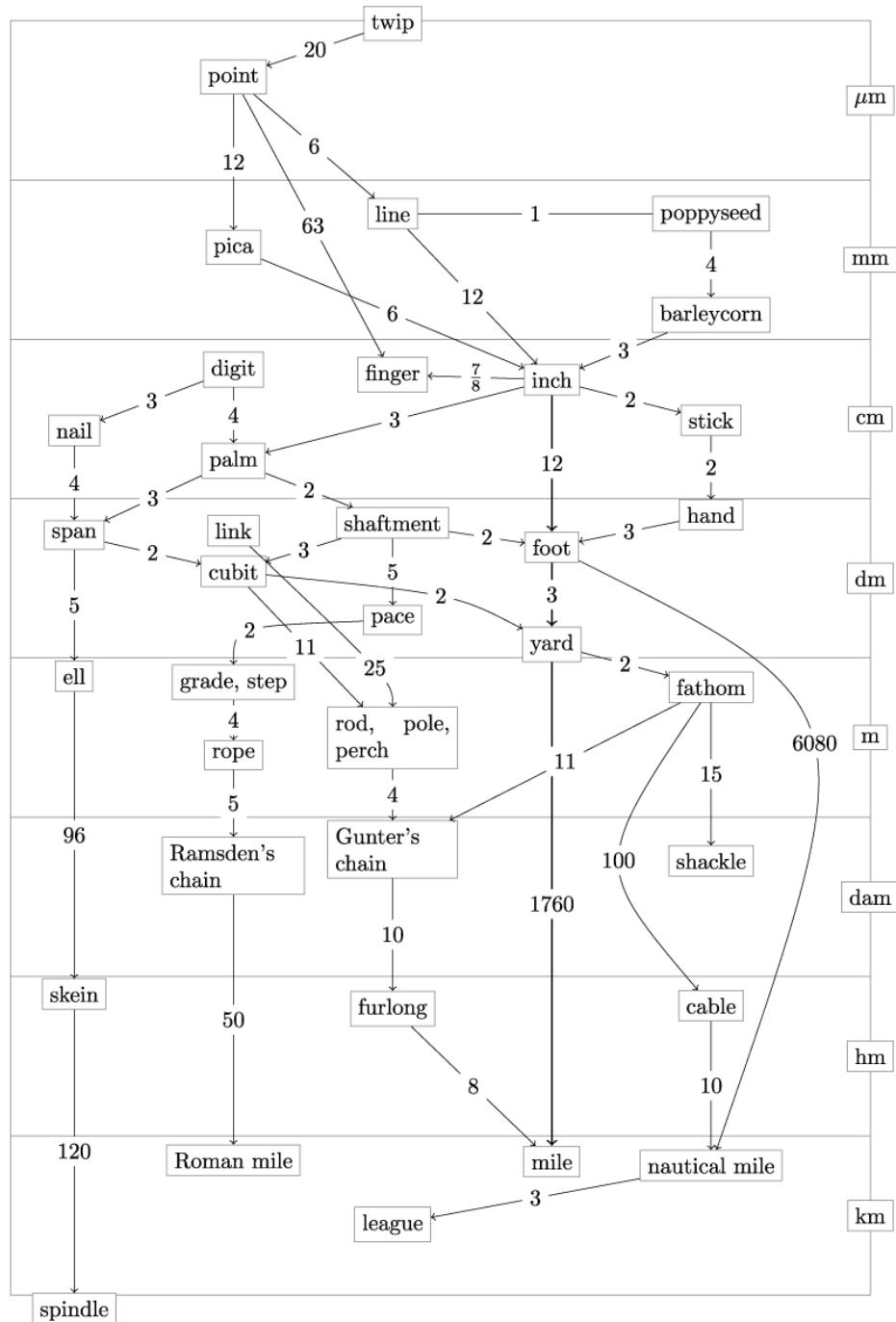


Image credit: 42CrMo4, Christoph Päper

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport()

## Synopsis

```
f:equal-lengths( $a as map(*)?,  
                  $b as map(*)?) as xs:boolean
```

## Description

Lengths (see *Section 5.2, “Lengths and units”*) are stored in maps and so are not directly comparable. This function returns true if two lengths are equal.

# f:generate-id

Generate a unique identifier for an element.

Function:{http://docbook.org/ns/docbook/functions}generate-id#1
{http://docbook.org/ns/docbook/functions}generate-id#2
Defined in modules/functions.xsl (2)
Used in: modules/functions.xsl, modules/unhandled.xsl, modules/index.xsl, modules/programming.xsl, modules/verbatim.xsl, modules/tablehtml.xsl, modules/xlink.xsl, modules/links.xsl, modules/attributes.xsl, modules/annotations.xsl, modules/chunk.xsl
Used by: f:generate-id(), f:generate-id(), f:unique-id(), f:href(), f:id(), f:chunk-filename()

## Synopsis

**f:generate-id(** \$node as element() **) as** xs:string

**f:generate-id(** \$node as element(),  
                  \$use-xml-id as xs:boolean) **) as** xs:string

## Description

This function returns a unique ID for an element. The ID is generated from the names of the ancestors and preceding siblings of the element. This value will be consistent across different transformations as long as the ancestors and preceding siblings remain unchanged.

If `use-xml-id` is true and there's an element with a `xml:id` attribute among the ancestors, the unique identifier is rooted at that element, rather than traversing all the way to the root of the document.

Disabling this feature by setting `use-xml-id` to false allows for the creation of globally unique IDs with `f:unique-id()`.

# f:gentext

Returns generated text.

## Description

Generated text now uses `m:gentext`. See *Chapter 4, Localization*.

# f:gentext-letters

Returns the letters for the in-scope language.

Function: {http://docbook.org/ns/docbook/functions}gentext-letters#1

Defined in: modules/functions.xsl

Used in: Not used.

## Synopsis

**f:gentext-letters(**`$node` as element()) as element(l:letters)

## Description

This function works calls `f:language()` on the specified node and then calls `f:gentext-letters-for-language()` with that language.

# f:gentext-letters-for-language

Returns the letters for a language.

Function: {http://docbook.org/ns/docbook/functions}gentext-letters-for-language#1

Defined in modules/functions.xsl

Used in: modules/functions.xsl, modules/index.xsl

Used by: f:gentext-letters()

## Synopsis

**f:gentext-letters-for-language( \$node ) as element() as element(l:letters)**

## Description

Returns a language-specific list of letters. These are used, for example, to construct the divisions in an index.

# f:highlight-verbatim

Syntax highlight verbatim element?

Function: {<http://docbook.org/ns/docbook/functions>}highlight-verbatim#1

Defined in: modules/verbatim.xsl

Used in: modules/verbatim.xsl

## Synopsis

**f:highlight-verbatim( \$node ) as element() as xs:boolean**

## Description

Returns true if the specified verbatim environment should have syntax highlighting applied to it. This depends on the `language` specified on the environment and the `$verbatim-syntax-highlight-languages` parameter.

# f:href

Returns the href link to a node.

Function:{http://docbook.org/ns/docbook/functions}href#2

Defined in modules/functions.xsl

Used in: modules/toc.xsl, modules/glossary.xsl, modules/index.xsl, modules/programming.xsl, modules/objects.xsl, modules/xlink.xsl, modules/links.xsl

Used by: t:index-zone-reference

## Synopsis

```
f:href($context as node(),
       $node as element()) as xs:string
```

## Description

This function generates a value for an HTML `href` attribute at the position of the context that will link to the node.

# f:id

Returns an ID for the element.

Function:{http://docbook.org/ns/docbook/functions}id#1

Defined in modules/functions.xsl

Used in: modules/toc.xsl, modules/blocks.xsl, modules/objects.xsl, modules/footnotes.xsl, modules/links.xsl

Used by: t:mediaobject-img

## Synopsis

`f:id( $node as element() ) as xs:string`

## Description

This function returns the ID of the element. If the element has an `xml:id` attribute, the value of that attribute is returned. Otherwise, the function constructs a unique ID value for the element.

The simplest way to construct a unique ID is with the `f:generate-id()` function. However, those values are only unique within the context of a single transformation and tend to vary over time. This can break anchors used in published documents and leads to spurious differences between documents.

The stylesheets generate their own unique IDs with `f:generate-id()`.

# f:in-scope-language

Returns the in-scope-language associated with an element.

Function: {http://docbook.org/ns/docbook/functions}in-scope-language#1

Defined in: modules/gentext.xsl

Used in: Not used.

Since: 2.0.0

## Synopsis

**f:in-scope-language**(`$target` as node()) as `xs:string`

## Description

Returns the value of the nearest in-scope `xml:lang` attribute or `$default-language` if no such attribute exists.

# f:intra-number-separator

Return the intra-number separator.

## Description

Separators are now part of the generated text template, see *Chapter 4, Localization*.

# f:is-empty-length

Tests if a length is empty.

Function: {http://docbook.org/ns/docbook/functions}is-empty-length#1

Defined in: modules/units.xsl

Used in: modules/units.xsl, modules/objects.xsl

Used by: f:equal-lengths(), f:mediaobject-viewport()

## Synopsis

`f:is-empty-length( $length as map(*)?) as xs:boolean`

## Description

Returns true if the length is empty. For a discussion of units and lengths, see *Section 5.2, “Lengths and units”*.

# f:is-true

Determines if a given value represents “true”

Function:{http://docbook.org/ns/docbook/functions}is-true#1

Defined in modules/functions.xsl

Used in: docbook.xsl, main.xsl, modules/variable.xsl, modules/head.xsl, modules/titles.xsl, modules/gentext.xsl, modules/toc.xsl, modules/index.xsl, modules/info.xsl, modules/blocks.xsl, modules/objects.xsl, modules/verbatim.xsl, modules/inlines.xsl, modules/xlink.xsl, modules/chunk-cleanup.xsl, modules/chunk-output.xsl

Used by: \$v:verbatim-number-first-line, \$v:chunk-renumber-footnotes, \$v:title-groups, t:generate-index, t:person-name-first-last, f:mediaobject-amend-uri(), f:verbatim-numbered(), f:verbatim-trim-trailing(), t:docbook

## Synopsis

`f:is-true( $value ) as xs:boolean`

## Description

There are several conventions for indicating “true” or “false”. XSLT uses the words “yes” and “no” in many places. The XML Schema data type for `xs:boolean` uses “true” and “false” but also “1” and “0”.

The `f:is-true()` function returns `true()` for any of “`true()`”, “yes”, “true”, or a value that can be cast to an integer if that integer is not zero. It returns `false()` for any of “`false()`”, “no”, “false”, a value that can be cast to an integer that is zero, or the empty sequence.

It reports an error for any other value and returns “`true()`”.

# f:l10n-language

Returns the best matching localization language.

Function:{http://docbook.org/ns/docbook/functions}l10n-language#1

Defined in modules/functions.xsl

Used in: modules/gentext.xsl, modules/l10n.xsl, modules/functions.xsl, modules/index.xsl, modules/inline.xsl

Used by: f:l10n-token(), f:gentext-letters-for-language(), t:generate-index

## Synopsis

`f:l10n-language( $target as element() ) as xs:string`

## Description

This function returns the localization language that best matches the language of the `$target` node. If `$gentext-language` is specified, it is always returned. The `$default-language` will be returned if there is no localization available for the declared language of the `$target`.

# f:l10n-token

Returns the gentext token for a key.

Function:{http://docbook.org/ns/docbook/functions}l10n-token#2

{http://docbook.org/ns/docbook/functions}l10n-token#3

Defined in modules/l10n.xsl (2)

Used in: modules/l10n.xsl, modules/titles.xsl, modules/gentext.xsl, modules/biblio690.xsl, modules/chunk-output.xsl

Since: 2.0.0

## Synopsis

```
f:l10n-token( $context as element(),
                $key as xs:string) as item()*
```

```
f:l10n-token( $context as element(),
                $lang as xs:string,
                $key as xs:string) as item()*
```

## Description

This function returns the gentext token for a key. This is usually accomplished by looking in the localization, but if you need an override for a specific element, you can replace this function.

# f:label-separator

Returns the label separator.

Function: {http://docbook.org/ns/docbook/functions}label-separator#1

Defined in: modules/functions.xsl

Used in: Not used.

## Synopsis

`f:label-separator( $node ) as element() as node()*`

## Description

The label separator separates a label from the number or title that follows it.

# f:language

Returns the language associated with an element.

## Description

Renamed `f:in-scope-language()`.

# f:languages

Returns all of the localizations used by a document.

Function: {http://docbook.org/ns/docbook/functions}languages#1

Defined in: modules/gentext.xsl

Used in: Not used.

Since: 2.0.0

## Synopsis

`f:languages( $context as document-node() ) as xs:string+`

## Description

Returns all of the localizations that might be used in formatting this document.

# f:length-string

Returns the string representation of a length.

Function: {http://docbook.org/ns/docbook/functions}length-string#1

Defined in: modules/units.xsl

Used in: Not used.

## Synopsis

**f:length-string( \$length as map(\*)?) as xs:string?**

## Description

For a given length (see *Section 5.2, “Lengths and units”*), returns the relative or absolute length formatted as a string. Relative lengths are formatted as the length followed by a literal “`*`” (e.g., “3\*”). Absolute lengths are formatted as the length followed by the unit (e.g., “2.54cm”).

# f:length-units

Returns the units associated with a length.

Function: {http://docbook.org/ns/docbook/functions}length-units#1

Defined in: modules/units.xsl

Used in: Not used.

## Synopsis

`f:length-units( $length as xs:string?) as xs:string?`

## Description

For a given length (see *Section 5.2, “Lengths and units”*), returns the units associated with the length. Returns an empty sequence for relative lengths.

# f:locales

Returns the locales detected in the document.

## Description

Replaced by `f:languages()`.

# f:make-length

Construct a length from constituent parts.

```
Function:{http://docbook.org/ns/docbook/functions}make-length#1
    {http://docbook.org/ns/docbook/functions}make-length#2
    {http://docbook.org/ns/docbook/functions}make-length#3
Defined in: modules/units.xsl (3)
Used in: modules/units.xsl, modules/objects.xsl
Used by: f:make-length(), f:mediaobject-viewport(), f:object-contentwidth(), f:object-contentheight()
```

## Synopsis

**f:make-length( \$relative as xs:double) as map(\*)**

**f:make-length( \$magnitude as xs:double,  
\$unit as xs:string) as map(\*)**

**f:make-length( \$relative as xs:double,  
\$magnitude as xs:double,  
\$unit as xs:string) as map(\*)**

## Description

These functions construct a length (see *Section 5.2, “Lengths and units”*) from either a relative magnitude, an absolute magnitude and a unit, or both.

# f:mediaobject-amend-uri

Amend the URI for media objects.

Function: {http://docbook.org/ns/docbook/functions}mediaobject-amend-uri#1

Defined in modules/objects.xsl

Used in: modules/objects.xsl

## Synopsis

`f:mediaobject-amend-uri($uri as xs:string) as xs:string`

## Description

After the media object URI has been calculated, `f:mediaobject-amend-uri()` is called. This is an opportunity to update the URI so that the media object will be found.

The default version of this function returns the URI unchanged unless `$mediaobject-grouped-by-type` is true. If media objects are grouped by type, the media object type is added to the URI. The media object type is determined by calling `f:mediaobject-type()`.

Suppose, for example, that the calculated URI is `file:///path/to/image.jpg` and the `f:mediaobject-type()` returns “`jpeg`”. In that case, the URI returned will be `file:///path/to/jpeg/image.jpg`.

### Important



Consider how *this* function interacts with the `m:mediaobject-output-adjust` mode. In particular, beware that the value processed in the `m:mediaobject-output-adjust` mode will *already* have been updated by `f:mediaobject-amend-uri()`.

# f:mediaobject-input-base-uri

Identify the input base URI for media.

Function: {http://docbook.org/ns/docbook/functions}mediaobject-input-base-uri#1

Defined in modules/objects.xsl

Used in: modules/objects.xsl

## Synopsis

**f:mediaobject-input-base-uri( \$node ) as element() as xs:string**

## Description

This value is calculated from the `$mediaobject-input-base-uri` parameter. If the parameter is the empty string, then this value is the empty sequence. Otherwise, this value is the absolute URI that results from resolving the value of the `$mediaobject-input-base-uri` parameter against the base URI of the input document.

In previous versions of the stylesheet, this value was computed once as a global variable. It was changed to a function because of failures resolving the base URI of documents when using XSpec.

# f:mediaobject-type

Return the type of a media object.

Function: {http://docbook.org/ns/docbook/functions}mediaobject-type#1

Defined in: modules/objects.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-amend-uri()

## Synopsis

**f:mediaobject-type( \$uri as xs:string) as xs:string?**

## Description

If `$mediaobject-grouped-by-type` is true, this function will be called to determine the type of each media object. The default version simply returns the media object extension, if there is one. (The type of `image.png` is `png`.)

# f:mediaobject-viewport

Construct the “viewport” for media objects.

Function: {http://docbook.org/ns/docbook/functions}mediaobject-viewport#1

Defined in modules/objects.xsl

Used in: modules/objects.xsl

## Synopsis

`f:mediaobject-viewport( $info ) as map(*) as map(*)`

## Description

If your DocBook markup for media objects includes any of the size or scaling adjustment attributes, this function is called to construct the HTML markup that implements those adjustments.

# f:number-separator

Returns the number separator.

## Description

Separators are now part of the generated text template, see *Chapter 4, Localization*.

# f:object-align

Returns the alignment of a media object.

Function: {http://docbook.org/ns/docbook/functions}object-align#1

Defined in: modules/objects.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport()

## Synopsis

**f:object-align( \$info ) as map(\*) as xs:string?**

## Description

By default, this function simply returns the value of the object's `align` attribute.

# f:object-contentheight

Returns the content height of an object.

Function: {<http://docbook.org/ns/docbook/functions>}object-contentheight#2

Defined in `modules/objects.xsl`

Used in: `modules/objects.xsl`

Used by: `f:mediaobject-viewport()`

## Synopsis

```
f:object-contentheight( $info as map(*),
                           $intrinsicheight as map(*)) as map(*)
```

## Description

Returns the content height, for reasons of historical accident, in the `contentdepth` attribute. This function returns the height of an object as a length (see *Section 5.2, “Lengths and units”*), if it has one. If the height is specified as a percentage, and the intrinsic size of the object is known, the an absolute length equal to that percentage of the intrinsic size is returned.

If `$image-ignore-scaling` is true, or no `contentdepth` is specified for the object, the empty sequence is returned.

# f:object-contentwidth

Returns the content width of an object.

Function: {<http://docbook.org/ns/docbook/functions>}object-contentwidth#2

Defined in: modules/objects.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport()

## Synopsis

```
f:object-contentwidth( $info as map(*),
                        $intrinsicwidth as map(*)) as map(*)
```

## Description

Returns the content width. This function returns the width of an object as a length (see *Section 5.2, “Lengths and units”*), if it has one. If the width is specified as a percentage, and the intrinsic size of the object is known, the an absolute length equal to that percentage of the intrinsic size is returned.

If `$image-ignore-scaling` is true, or no `contentwidth` is specified for the object, the empty sequence is returned.

# f:object-height

Returns the height (depth) of a `mediaobject`.

Function: `{http://docbook.org/ns/docbook/functions}object-height#1`

Defined in: `modules/objects.xsl`

Used in: `modules/objects.xsl`

Used by: `f:mediaobject-viewport(), f:object-scalefit()`

## Synopsis

`f:object-height( $info as map(*) ) as map(*)`

## Description

The height of a `mediaobject` is specified, for reasons of historical accident, in the `depth` attribute. This function returns the height of an object as a length (see *Section 5.2, “Lengths and units”*), if it has one. If `$image-ignore-scaling` is true, or no `depth` is specified for the object, the empty sequence is returned.

# f:object-properties

Returns the properties of an object.

Function: {http://docbook.org/ns/docbook/functions}object-properties#1

{http://docbook.org/ns/docbook/functions}object-properties#2

Defined in: modules/objects.xsl (2)

Used in: modules/objects.xsl

Used by: f:object-properties()

## Synopsis

**f:object-properties( \$uri as xs:string) as map(xs:string, xs:anyAtomicType)**

**f:object-properties( \$uri as xs:string,  
\$image-file as xs:boolean) as map(xs:string, xs:anyAtomicType)**

## Description

If the extension functions are available, the stylesheets will interrogate images for their properties. The goal is to return, at a minimum, the height and width of the image. This is used to compute viewport and scaling factors. If the image metadata extension is available, then considerably more information may be returned in the map.

For example, the image metadata properties of the Amaryllis photograph in *Figure 1, “An amaryllis”* are:

Property name	Property value <sup>a</sup>
apple-multi-language-profile-name	38 hrHR(LCD u boji) koKR(컬러 LCD) nb...
background-color	R 255, G 255, B 255
bits-per-sample	8
blue-colorant	(0.1431, 0.0606, 0.7141)
blue-parametric-trc	para (0x70617261): 32 bytes
blue-trc	0.0, 0.0000763, 0.0001526, 0.000228...
blue-x	15000
blue-y	6000
class	Display Device
cmm-type	Lino

<b>Property name</b>	<b>Property value<sup>a</sup></b>
color-space	RGB
color-type	True Color with Alpha
component-1	Y component: Quantization table 0, ...
component-2	Cb component: Quantization table 1,...
component-3	Cr component: Quantization table 1,...
compression-type	Baseline
data-precision	8 bits
detected-file-type-long-name	Joint Photographic Experts Group
detected-file-type-name	JPEG
detected-mime-type	image/jpeg
device-manufacturer	IEC
device-mfg-description	IEC http://www.iec.ch
device-model	sRGB
device-model-description	IEC 61966-2.1 Default RGB colour sp...
exif-image-height	1868 pixels
exif-image-width	1516 pixels
expected-file-name-extension	jpg
filter-method	Adaptive
green-colorant	(0.3851, 0.7169, 0.0971)
green-parametric-trc	para (0x70617261): 32 bytes
green-trc	0.0, 0.0000763, 0.0001526, 0.000228...
green-x	30000
green-y	60000
height	256
icc-profile-name	icc
image-gamma	0.455
image-height	336 pixels
image-width	500 pixels
interlace-method	No Interlace
last-modification-time	2022-12-30T15:03:35
luminance	(76.0365, 80, 87.1246)
make-and-model	mmod (0x6D6D6F64): 40 bytes
measurement	1931 2° Observer, Backing (0, 0, 0)...
media-black-point	(0, 0, 0)
media-white-point	(0.9505, 1, 1.0891)
native-display-information	ndin (0x6E64696E): 62 bytes
number-of-components	3
number-of-tables	4 Huffman tables
orientation	Top, left side (Horizontal / normal...

<b>Property name</b>	<b>Property value <sup>a</sup></b>
pixels-per-unit-x	5669
pixels-per-unit-y	5669
primary-platform	Microsoft Corporation
profile-connection-space	XYZ
profile-copyright	Copyright (c) 1998 Hewlett-Packard ...
profile-date/time	1998-02-09T06:49:00
profile-description	sRGB IEC61966-2.1
profile-size	3144
red-colorant	(0.4361, 0.2225, 0.0139)
red-parametric-trc	para (0x70617261): 32 bytes
red-trc	0.0, 0.0000763, 0.0001526, 0.000228...
red-x	64000
red-y	33000
resolution-unit	Inch
resolution-units	inch
signature	acsp
tag-count	17
technology	CRT
textual-data	xmp:UserComment: Screenshot
thumbnail-height-pixels	0
thumbnail-width-pixels	0
unit-specifier	Metres
unknown-tag-(0x76636770)	vcgp (0x76636770): 56 bytes
user-comment	Screenshot
version	2.1.0
video-card-gamma	vcgt (0x76636774): 48 bytes
viewing-conditions	view (0x76696577): 36 bytes
viewing-conditions-description	Reference Viewing Condition in IEC6...
white-point-x	31270
white-point-y	32900
width	400
x-resolution	72 dots
xyz-values	0.964 1 0.825
y-resolution	72 dots

<sup>a</sup>Some values have been truncated to prevent the table from becoming unwieldy. These values can be identified by a trailing ellipsis (...).



**Figure 1. An amaryllis**

# f:object-scale

Returns the scaling factor for an object.

Function: {http://docbook.org/ns/docbook/functions}object-scale#1

Defined in: modules/objects.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport()

## Synopsis

**f:object-scale( \$info as map(\*) ) as xs:double**

## Description

Returns the scaling factor for an object. Scaling only applies if `$image-ignore-scaling` is false and the object has a `scale` attribute.

# f:object-scalefit

Scale object to fit?

Function: {http://docbook.org/ns/docbook/functions}object-scalefit#1

Defined in: modules/objects.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport()

## Synopsis

**f:object-scalefit( \$info as map(\*) ) as xs:boolean**

## Description

This function determines whether or not an object should be scaled to fit its container.

# f:object-valign

Returns the vertical alignment of a media object.

Function: {http://docbook.org/ns/docbook/functions}object-valign#1

Defined in: modules/objects.xsl

Used in: modules/objects.xsl

Used by: f:mediaobject-viewport()

## Synopsis

`f:object-valign( $info as map(*) ) as xs:string?`

## Description

By default, this function simply returns the value of the object's `valign` attribute. If it doesn't have a `valign` attribute, the value "middle" is returned, because that's what previous versions of stylesheets for DocBook did.

# f:object-width

Returns the width of a `mediaobject`.

Function: `{http://docbook.org/ns/docbook/functions}object-width#1`

Defined in: `modules/objects.xsl`

Used in: `modules/objects.xsl`

Used by: `f:mediaobject-viewport(), f:object-scalefit()`

## Synopsis

`f:object-width( $info as map(*) ) as map(*)`

## Description

This function returns the width of an object as a length (see *Section 5.2, “Lengths and units”*), if it has one. If `$image-ignore-scaling` is true, or no `width` is specified for the object, the empty length is returned.

# f:orderedlist-item-number

Returns the item number of an item in an ordered list.

Function: {http://docbook.org/ns/docbook/functions}orderedlist-item-number#1

Defined in modules/functions.xsl

Used in: modules/titles.xsl, modules/functions.xsl

Used by: f:orderedlist-item-numeration()

## Synopsis

**f:orderedlist-item-number( \$node ) as element(db:listitem) as xs:integer+**

## Description

Returns the item number of a list item. This is always an integer, even if it will be formatted as a letter, roman numeral, or other symbol. For nested lists, this function returns the fully qualified item number. For example, for the second item in the fourth item in the third item of a list, it will return (2, 4, 3).

# f:orderedlist-item-numeration

Returns the item enumeration format for an ordered list.

Function:{http://docbook.org/ns/docbook/functions}orderedlist-item-numeration#1

Defined in modules/functions.xsl

Used in: modules/titles.xsl, modules/lists.xsl

## Synopsis

**f:orderedlist-item-numeration( \$node as element(db:listitem) ) as xs:string**

## Description

Computes the enumeration for the specified list item. See  
\$orderedlist-item-numeration .

# f:orderedlist-startingnumber

Returns the starting number of an ordered list.

Function: {http://docbook.org/ns/docbook/functions}orderedlist-startingnumber#1

Defined in modules/functions.xsl

Used in: modules/functions.xsl, modules/lists.xsl

Used by: f:orderedlist-startingnumber(), f:orderedlist-item-number()

## Synopsis

`f:orderedlist-startingnumber( $list as element(db:orderedlist) ) as xs:integer`

## Description

In most `orderedlist` elements, the first item in the list is item number “1”. However, the `startingnumber` attribute can specify a different initial number and, in the case of continued lists, the starting number depends on the ending number of the preceding list.

The `f:orderedlist-startingnumber()` returns the number of the first list item in an ordered list.

# f:parse-length

Parse a string into a length.

Function:{http://docbook.org/ns/docbook/functions}parse-length#1

Defined in modules/units.xsl

Used in: modules/variable.xsl, modules/units.xsl, modules/objects.xsl, modules/tablecals.xsl

Used by: \$v:nominal-page-width, \$v:image-nominal-width, \$v:image-nominal-height, f:length-units()

## Synopsis

`f:parse-length( $length as xs:string?) as map(*)`

## Description

This function parses a string such as “4in” or “3.14cm” or “50%” into a length. It will fall back to a distance of the `$default-length-magnitude` and `$default-length-unit` if the string cannot be parsed as a length.

# f:pi

Returns DocBook processing-instruction property values.

Function:{http://docbook.org/ns/docbook/functions}pi#2

{http://docbook.org/ns/docbook/functions}pi#3

Defined in modules/functions.xsl (2)

Used in: modules/functions.xsl, modules/bibliography.xsl, modules/index.xsl, modules/lists.xsl, modules(blocks.xsl, modules/programming.xsl, modules/objects.xsl, modules/verbatim.xsl, modules/xlink.xsl, modules/info.xsl, modules/chunk.xsl

Used by: f:pi(), f:verbatim-style(), f:verbatim-highlight(), f:xlink-style(), f:date-format(), f:verbatim-numbered(), f:verbatim-trim-trailing(), f:chunk-filename()

## Synopsis

```
f:pi( $context as node()?,  
      $property as xs:string) as xs:string?
```

```
f:pi( $context as node()?,  
      $property as xs:string,  
      $default as xs:string*) as xs:string*
```

## Description

Several DocBook elements have alternate presentations that can be specified with global parameters. Many of them can also be set on a per-element basis with a `db` processing instruction. The date format, for example, can be changed in this way as described in the `f:date-format()` function.

The `f:pi()` function takes a context and the name of a property. It returns the value specified for that property from all of the `db` processing instructions in the specified context. The three argument version allows you to specify a default value. It will be returned if there are no values specified for the property in that context.

# f:pi-attributes

Returns processing-instruction pseudo-attributes as attributes.

Function:{http://docbook.org/ns/docbook/functions}pi-attributes#1

Defined in modules/functions.xsl

Used in: modules/functions.xsl, modules/blocks.xsl, modules/objects.xsl, modules/verbatim.xsl, modules/tablecals.xsl, modules/inlines.xsl

## Synopsis

f:pi-attributes( \$pis as processing-instruction()\*) as element()?

## Description

Using pseudo-attributes in a db processing instruction is a convenient mechanism for providing additional options to the stylesheets. But processing them is tedious. This function gathers together the pseudo-attributes specified in a series of processing instructions and returns them as attributes on an element. If the same property occurs more than once in the sequence, the last value will be returned.

# f:post-label-punctuation

Punctuation that follows a label.

## Description

Generated text has been reworked, see *Chapter 4, Localization*.

# f:refsection

Is this a section in a `refentry`?

Function: `{http://docbook.org/ns/docbook/functions}refsection#1`

Defined in: `modules/functions.xsl`

Used in: `modules/functions.xsl`

Used by: `f:section()`

## Synopsis

`f:refsection( $node ) as element() as xs:boolean`

## Description

Returns true if the node is a section in a `refentry`, a `refsection`, `refsect1`, `refsect2`, or `refsect3` element.

# f:relative-length

Returns the relative portion of a length, if it has one.

Function: {http://docbook.org/ns/docbook/functions}relative-length#1

Defined in: modules/units.xsl

Used in: modules/tablecals.xsl

## Synopsis

**f:relative-length( \$length ) as map(\*) as xs:double**

## Description

For a given length, returns the relative length if it has one. For lengths that have no relative component, returns 0.0. For a discussion of units, see *Section 5.2, “Lengths and units”*.

# f:relative-path

Give two file paths, what's the relative path between them?

Function: {http://docbook.org/ns/docbook/functions}relative-path#2

Defined in: modules/functions.xsl

Used in: modules/objects.xsl

## Synopsis

```
f:relative-path( $base as xs:string,  
                  $path as xs:string) as xs:string
```

## Description

This function attempts to determine the relative path between two absolute paths. For example, given a base path of `/a/base/uri/path/` and a path of `/a/base/uri/with/other`, the results is `./with/other`.

If the path is an absolute URI (that isn't a `file:` URI), then the URI is returned unchanged.

# f:section

Is this a section?

Function:	{http://docbook.org/ns/docbook/functions}section#1
Defined in:	modules/functions.xsl
Used in:	Not used.

## Synopsis

**f:section( \$node ) as element() as xs:boolean**

## Description

Returns true if the node is a section, a `section` or `sect1-5` element.

# f:section-depth

Returns the section depth of a node.

Function: {http://docbook.org/ns/docbook/functions}section-depth#1

Defined in: modules/functions.xsl

Used in: modules/functions.xsl

Used by: f:section-depth()

## Synopsis

**f:section-depth( \$node ) as element()?) as xs:integer**

## Description

Returns the section depth of a node, that is, how many section elements are among its ancestors.

# f:spaces

Returns a string of spaces.

Function: {http://docbook.org/ns/docbook/functions}spaces#1

Defined in: modules/functions.xsl

Used in: modules/functions.xsl, modules/programming.xsl

Used by: f:spaces()

## Synopsis

`f:spaces( $length as item()*) as xs:string?`

## Description

This function returns a string of `$length` spaces. If `$length` :

- is empty, an empty sequence is returned.
- is an integer, or castable to an integer, a string of that length is returned.  
If the length is negative, an empty sequence is returned.
- is anything else, the length of the string value of `$length` is used as the length.

In other words:

- `f:spaces()` is ‘`()`’,
- `f:spaces(0)` is “(no spaces),
- `f:spaces(-1)` is ‘`()`’,
- `f:spaces(2)` is ‘’ (two spaces),
- `f:spaces("1")` is ‘’ (one space), and
- `f:spaces("test")` is ‘’ (four spaces).

This function accepts a sequence to support the empty sequence. If it's passed a sequence of more than one item, it treats that as if the string values of the items had been concatenated together. This can have slightly surprising consequences as “`('a', '3', '0')`” will return a string of three spaces whereas “`('0', '3', '0')`” will return a string of thirty.

# f:step-number

Returns the step number of a step in an procedure.

Function:	{http://docbook.org/ns/docbook/functions}step-number#1
Defined in:	modules/functions.xsl
Used in:	modules/functions.xsl
Used by:	f:step-numeration()

## Synopsis

**f:step-number( \$node as element(db:step)) as xs:integer+**

## Description

Returns the step number of step in a procedure. This is always an integer, even if it will be formatted as a letter, roman numeral, or other symbol. For steps nested in side `substeps`, this function returns the fully qualified item number. For example, for the second substep in the fourth substep in the third step of a procedure, it will return `(2, 4, 3)`.

# f:step-numeration

Returns the step numeration format for a step.

Function: {http://docbook.org/ns/docbook/functions}step-numeration#1

Defined in: modules/functions.xsl

Used in: modules/titles.xsl, modules/lists.xsl

## Synopsis

```
f:step-numeration( $node as element(db:step)) as xs:string
```

## Description

Computes the numeration for the specified step. See [\\$procedure-step-numeration](#).

# f:syntax-highlight

Performs syntax highlighting.

Function: {http://docbook.org/ns/docbook/functions}syntax-highlight#1

{http://docbook.org/ns/docbook/functions}syntax-highlight#2

{http://docbook.org/ns/docbook/functions}syntax-highlight#3

Defined in: highlight.xsl (3)

Used in: modules/verbatim.xsl, highlight.xsl

Used by: f:syntax-highlight(), f:syntax-highlight()

## Synopsis

`f:syntax-highlight( $source as xs:string)`

`f:syntax-highlight( $source as xs:string,  
$language as xs:string)`

`f:syntax-highlight( $source as xs:string,  
$options ,  
$pyoptions as map(xs:string, xs:string)) as node()*`

## Description

Verbatim environments that specify a `language` can be syntax highlighted. Syntax highlighting adds additional spans so that, for example, CSS can be used to make keyword bold or quoted strings a different color.

Syntax highlighting requires an extension function. The list of languages that will be syntax highlighted is taken from `$verbatim-syntax-highlight-languages`.

# f:target

Returns the element identified by a particular id.

Function: {http://docbook.org/ns/docbook/functions}target#2

Defined in: modules/functions.xsl

Used in: modules/programming.xsl, modules/links.xsl

## Synopsis

```
f:target( $id as xs:string,  
           $context as node() ) as element()*
```

## Description

The item with the specified ID that is in the same document as `$context` item is returned.

# f:template

Returns the title page template for an element.

Function:	{http://docbook.org/ns/docbook/functions}template#2
Defined in:	modules/templates.xsl
Used in:	modules/templates.xsl
Used by:	t:biblioentry

## Synopsis

```
f:template( $context as element(),
             $default as element() ) as element()
```

## Description

Returns the title page template that should be used for an element. If no template is found, returns the `$default` template.

# f:tokenize-on-char

Tokenize a string at a specific character.

Function: {http://docbook.org/ns/docbook/functions}tokenize-on-char#2

Defined in: modules/functions.xsl

Used in: modules/tablecals.xsl

## Synopsis

```
f:tokenize-on-char( $string as xs:string,  
                     $char as xs:string) as xs:string*
```

## Description

This function tokenizes a string at a specific character (not a regular expression). If the specified character is a regular expression meta-character, it is automatically escaped.

# f:unique-id

Returns an ID for the element.

Function: {http://docbook.org/ns/docbook/functions}unique-id#1

Defined in: modules/functions.xsl

Used in: Not used.

## Synopsis

`f:unique-id( $node as element() ) as xs:string`

## Description

This function returns a globally unique ID for the element. This allows a transformation to disambiguate between different elements that happen to have the same `xml:id`, as can occur if transclusion is performed.

# f:uri-scheme

Returns the scheme portion of a URI.

Function: {http://docbook.org/ns/docbook/functions}uri-scheme#1

Defined in modules/functions.xsl

Used in: modules/functions.xsl, modules/objects.xsl, modules/chunk-cleanup.xsl

Used by: f:relative-path(), f:mediaobject-amend-uri()

Since: 2.0.6

## Synopsis

**f:uri-scheme( \$uri as xs:string) as xs:string?**

## Description

If the URI begins with a scheme, this function returns the scheme. Otherwise, it returns an empty sequence.

# f:verbatim-highlight

Returns the highlight properties for a verbatim environment.

Function: {<http://docbook.org/ns/docbook/functions>}verbatim-highlight#1

Defined in: modules/verbatim.xsl

Used in: modules/verbatim.xsl

## Synopsis

`f:verbatim-highlight( $context ) as element() as xs:string*`

## Description

Verbatim elements can be styled in a variety of ways, see *Section 5.3, “Verbatim styles”*. This function returns the highlight settings for a particular verbatim environment.

If the style for this environment is explicitly “plain”, then no highlight settings are returned (this avoids a bunch of warning messages). Otherwise, if a `db` processing instruction with a `verbatim-highlight` pseudo-attribute is present, it is returned. If there’s no PI, then the default settings for this kind of environment are returned.

# f:verbatim-numbered

Returns true if the listing should be numbered.

Function: {<http://docbook.org/ns/docbook/functions>}verbatim-numbered#1

Defined in: modules/verbatim.xsl

Used in: modules/verbatim.xsl, modules/attributes.xsl

## Synopsis

**f:verbatim-numbered(** \$context **as element()) as xs:boolean**

## Description

The `f:verbatim-numbered()` function returns true if the specified verbatim listing should be numbered. This will be true if:

- The `linenumbering` attribute is specified and has the value `numbered`,
- the element's name is listed in `$verbatim-numbered-elements`, or
- a `db` processing instruction with a `verbatim-numbered` pseudo-attribute that is true is specified.

There are two overriding conditions that will force the value to be false: first, if the `linenumbering` attribute is specified and has the value `unnumbered`, the listing will not be numbered. Second if the listing has a `db` processing instruction with a `verbatim-style` pseudo attribute that specifies the “plain” style. Listings in the plain style cannot be numbered, attempting to do so will generate a warning message from the stylesheets.

# f:verbatim-style

Returns the style of a verbatim listing.

Function: {http://docbook.org/ns/docbook/functions}verbatim-style#1

Defined in: modules/verbatim.xsl

Used in: modules/verbatim.xsl, modules/attributes.xsl

## Synopsis

`f:verbatim-style( $context as element() ) as xs:string`

## Description

The `f:verbatim-style()` function returns the style of presentation that should be used for the specified verbatim listing.

If a `db` processing instruction with a `verbatim-style` pseudo-attribute is present, the value of that pseudo-attribute will be returned. Otherwise, it will be “`lines`” if the element’s name is listed in `$verbatim-line-style`, “`plain`” if the element’s name is listed in `$verbatim-plain-style`, or “`raw`”.

# f:verbatim-trim-trailing

Return true if trailing blank lines should be trimmed.

Function: {http://docbook.org/ns/docbook/functions}verbatim-trim-trailing#1

Defined in modules/verbatim.xsl

Used in: modules/verbatim.xsl

## Synopsis

`f:verbatim-trim-trailing( $context as element() ) as xs:boolean`

## Description

This function examines a verbatim environment, the `$context`, and returns true if trailing blank lines should be trimmed from it. A `db` processing instruction with a `verbatim-trim-trailing` pseudo-attribute may be used to override the `$verbatim-trim-trailing-blank-lines` parameter on a case-by-case basis.

# f:xlink-style

Returns the XLink style for this document.

Function: {<http://docbook.org/ns/docbook/functions>}xlink-style#1

Defined in: modules/xlink.xsl

Used in: main.xsl, modules/xlink.xsl

## Synopsis

**f:xlink-style( \$document ) as document-node() as xs:string**

## Description

The stylesheets support XLink extended links. See [\\$xlink-style](#) for more details.

# f:xpointer-idref

Parses an XPointer for a same-document ID reference.

Function: {http://docbook.org/ns/docbook/functions}xpointer-idref#1

Defined in: modules/xlink.xsl

Used in: modules/xlink.xsl

## Synopsis

`f:xpointer-idref( $xpointer ) as xs:string?`

## Description

In DocBook, there are generally two ways to refer to other elements in the current document by ID: using the `linkend` (or `linkends`) attribute or using the `xlink:href` attribute.

The `f:xpointer-idref()` function returns the ID portion of an XPointer that is a reference to another ID in the current document.

In other words:

- `f:xpointer-idref('#foo')` is `foo`,
- `f:xpointer-idref('id(foo)')` is `foo`,
- and `f:xpointer-idref('http://example.org/#foo')` is the empty sequence.



# IV. Template reference

# ~~t:audio-fallback~~

Generate fallback for audio.

## Description

Audio fallback is now supported with `$fallback-js`.

# t:biblioentry

Format a `biblioentry` element.

Template: {http://docbook.org/ns/docbook/templates}biblioentry

Defined in: modules/templates.xsl

Used in: modules/bibliography.xsl

## Synopsis

```
<xsl:template name="t:biblioentry"/>
```

## Description

Unlike `bibliomixed`, where the author is expected to provide most of the formatting, `biblioentry` is a bag of metadata. On the one hand, this opens up the possibility of producing different bibliographic styles from the same entry. On the other hand, it greatly complicates formatting.

The default `t:biblioentry` template uses a header template (see *Section 5.5, “Templates”*).

# t:bottom-nav

Generate bottom-of-page navigation links.

Template: {http://docbook.org/ns/docbook/templates}bottom-nav

Defined in: modules/chunk-cleanup.xsl

Used in: modules/chunk-cleanup.xsl

## Synopsis

```
<xsl:template name="t:bottom-nav">
    <xsl:param name="chunk" as="xs:boolean"/>
    <xsl:param name="node" as="element()"/>
    <xsl:param name="prev" as="element()?">
    <xsl:param name="next" as="element()?">
    <xsl:param name="up" as="element()?">
    <xsl:param name="top" as="element()?">
</xsl:template>
```

## Description

This template generates the bottom-of-page `nav` element. When chunking is performed, this should provide links between pages.

# t:chunk-cleanup

Cleanup the HTML markup for a DocBook document.

Template:	{http://docbook.org/ns/docbook/templates}chunk-cleanup
Defined in:	docbook.xsl
Used in:	docbook.xsl
Used by:	t:docbook

## Synopsis

```
<xsl:template name="t:chunk-cleanup">
    <xsl:param name="source" as="document-node()"/>
    <xsl:param name="docbook" as="document-node()"/>
</xsl:template>
```

## Description

This mode performs cleanup tasks on the HTML document that results from DocBook processing. This process moves footnotes and annotations when necessary, generates navigation elements, etc.

# t:chunk-footnotes

Processes footnotes in a chunk.

Template: {http://docbook.org/ns/docbook/templates}chunk-footnotes

Defined in: modules/chunk-cleanup.xsl

Used in: modules/chunk-cleanup.xsl

## Synopsis

```
<xsl:template name="t:chunk-footnotes">
    <xsl:param name="footnotes" as="element()*"/>
</xsl:template>
```

## Description

When chunking is being performed (see *Section 2.6, ““Chunked” output”*), all of the footnotes that should appear in a chunk are processed with this template.

# t:chunk-output

Generate chunked output for a DocBook document.

Template:	{http://docbook.org/ns/docbook/templates}chunk-output
Defined in:	docbook.xsl
Used in:	docbook.xsl
Used by:	t:docbook

## Synopsis

```
<xsl:template name="t:chunk-output">
    <xsl:param name="source" as="document-node()"/>
    <xsl:param name="docbook" as="document-node()"/>
</xsl:template>
```

## Description

This mode processes the `$persistent-toc` if requested and generates individual documents for each chunk. Like the XPath `transform` function, this template returns a map. You must process the map to store the documents.

# t:docbook

Process a DocBook document.

Template:	{http://docbook.org/ns/docbook/templates}docbook
Defined in:	docbook.xsl
Used in:	docbook.xsl

## Synopsis

```
<xsl:template name="t:docbook">
    <xsl:param name="return" as="xs:string"/>
</xsl:template>
```

## Description

This mode processes a DocBook document. It's usually called automatically by the root template in the stylesheets. You only need to call it explicitly if you have an override for the root template and want to do some specialized processing.

The template has a single parameter, `return`, which selects what should be constructed by the template:

`main-document`, the default value

Returns the primary output of the stylesheet.

`raw-results`

Returns a map with two keys, `document` containing the original DocBook document processed and `output` containing the transformed result.

`chunked-results`

Returns a map with two keys, `document` containing the original DocBook document processed and `chunks` containing a map from output URIs to documents.

Any other value is an error.

# t:generate-index

Generate a back-of-the-book style index.

Template: {http://docbook.org/ns/docbook/templates}generate-index

Defined in: modules/index.xsl

Used in: modules/index.xsl

## Synopsis

```
<xsl:template name="t:generate-index">
    <xsl:param name="scope"/>
</xsl:template>
```

## Description

This template is called when an empty `index` tag is found the document and automatic index generation is enabled. Automatic index generation is enabled if either the `$generate-index` is true or a `db` processing instruction with an `autoindex` pseudo-attribute is true. If the `autoindex` pseudo-attribute is explicitly false, automatic index generation is suppressed for the index in which it appears.

# t:index-zone-reference

Process index zone references.

Template: {http://docbook.org/ns/docbook/templates}index-zone-reference

Defined in: modules/index.xsl

Used in: modules/index.xsl

Used by: t:index-zone-reference

## Synopsis

```
<xsl:template name="t:index-zone-reference">
    <xsl:param name="scope"/>
    <xsl:param name="role"/>
    <xsl:param name="type"/>
    <xsl:param name="lang"/>
    <xsl:param name="zones" as="xs:string*"/>
</xsl:template>
```

## Description

This template is called to generate index references for terms that have a `zone` attribute. It is only called when generating an index.

# t:inline

Process an inline element.

Template:{http://docbook.org/ns/docbook/templates}inline

Defined in modules/inlines.xsl

Used in: modules/bibliography.xsl, modules/glossary.xsl, modules/info.xsl,  
modules(blocks.xsl, modules/verbatim.xsl, modules/inlines.xsl, modules/  
links.xsl, modules/publishers.xsl

## Synopsis

```
<xsl:template name="t:inline">
    <xsl:param name="namemap"/>
    <xsl:param name="class" as="xs:string*" />
    <xsl:param name="local-name-as-class" as="xs:boolean"/>
    <xsl:param name="extra-attributes" as="attribute()*"/>
    <xsl:param name="content"/>
</xsl:template>
```

## Description

All of the inline elements are processed by `t:inline`. This provides a common place to handle the fact that any of them might be a link. Several parameters exist to handle variations in presentation:

`namemap`

Maps the element to a particular HTML element. Defaults to `span`.

`class`

A list of tokens that should be added to the element's `class` attribute.

`local-name-as-class`

If this is true, the local name of the DocBook element being processed will be added to the classes.

`content`

The element content. By default, this is obtained by applying XSLT templates to the element's content.

`extra-attributes`

Extra attributes to be added to the element.

# t:mediaobject-img

Generate a mediaobject img.

Template: {http://docbook.org/ns/docbook/templates}mediaobject-img

Defined in: modules/objects.xsl

Used in: modules/objects.xsl

## Synopsis

```
<xsl:template name="t:mediaobject-img">
    <xsl:param name="filename" as="xs:string"/>
    <xsl:param name="styles" as="xs:string*" />
    <xsl:param name="viewport" as="map(*)?" />
    <xsl:param name="imageproperties" as="map(*)?" />
</xsl:template>
```

## Description

This template is called to generate the `img` element for a media object.

# t:person-name

Formats a personal name.

Template: {http://docbook.org/ns/docbook/templates}person-name

Defined in: modules/info.xsl

Used in: modules/info.xsl

## Synopsis

```
<xsl:template name="t:person-name">
    <xsl:param name="style" as="xs:string"/>
</xsl:template>
```

## Description

This template formats a personal name according to the specified style. This template must only be called when the current context item is a `personname`. The `personname` element can be used in two different ways, it can just contain the name:

```
<personname>Norman Walsh</personname>
```

or it can contain the name with markup:

```
<personname><givenname>Norman</givenname>
<surname>Walsh</surname></personname>
```

How the name is formatted depends on which form of markup is used and which style is selected. Several styles are supported:

1. If the name contains no markup, it is simply formatted as is.
2. If the style is “`FAMILY-given`”, it is formatted with `t:person-name-family-given` and generally produces the family name in upper case followed by the given name. For example: “WALSH Norman”.
3. If the style is “`last-first`”, it is formatted with `t:person-name-last-first` and generally produces the family name followed by the given name separated by a comma. For example: “Walsh, Norman”.

4. If the style is “first-last”, it is formatted with `t:person-name-first-last` and generally produces the given name followed by the family name. For example: “Norman Walsh”.
5. Any other value raises `dbe:INVALID-NAME-STYLE`.

### Given names



Historically, DocBook used `firstname` for the given name of a person. In DocBook 5.1, the `givenname` element was introduced as an alternative. The stylesheets treat them identically.

The stylesheets attempt to determine what style to use for each `personname`:

1. First, by looking at that the `role` attribute on that element. If any of styles from the `$v:personal-name-styles` list appear in the role, that style will be selected. (If more than one style appears, the results are undefined.)
2. If the `personname` doesn't identify a style, and if it's a child of `author`, `editor`, or `othercredit`, the `role` attribute of its parent is inspected.
3. If the `$personal-name-style` is defined, it is selected.
4. If a style still hasn't been selected the style in the localization file is used.

Personal names are notoriously complicated (see *Names*). DocBook includes several tags for identifying parts of names:

`firstname` or `givenname`

for a given name,

`surname`

for a family name,

`lineage`

for lineage (such as “Jr” or “III”),

`honorific`

for an honorific title (such as “Dr” or “Ms”), and

othername

for everything else.

In all likelihood, if your document contains a variety of personal names, you will need to use roles to disambiguate special cases and you will have to provide alternate or additional templates to format them.

# t:person-name-family-given

Formats a personal name in the “FAMILY-given” style.

Template: {http://docbook.org/ns/docbook/templates}person-name-family-given  
Defined in modules/info.xsl  
Used in: modules/info.xsl  
Used by: t:person-name

## Synopsis

```
<xsl:template name="t:person-name-family-given"/>
```

## Description

The “FAMILY-given” style formats a personal name as the family name (surname) in upper-case, followed by the first given name (if there is one), followed by the text “[FAMILY given]”. See t:person-name.

*Example 1, “The FAMILY-given name style”* shows an example of a personal name to be formatted with the “FAMILY-given” style.

```
<para>The author is  
<personname role="FAMILY-given">  
    <givenname>Norman</givenname>  
    <surname>Walsh</surname>  
    <othername>David</othername>  
</personname>.</para>
```

### Example 1. The FAMILY-given name style

That would be formatted as:

The author is WALSH Norman [FAMILY Given].

# t:person-name-first-last

Formats a personal name in the “first-last” style.

Template: {http://docbook.org/ns/docbook/templates}person-name-first-last

Defined in modules/info.xsl

Used in: modules/info.xsl

Used by: t:person-name

## Synopsis

```
<xsl:template name="t:person-name-first-last"/>
```

## Description

The “first-last” style formats a personal name as: the first honorific , followed by the given name, followed possibly by the “middle name”, followed by the surname , followed by the lineage . See t:person-name .

If an othername is provided and if \$othername-in-middle is true, then the first othername will be treated as a “middle name” and presented between the given and family names.

*Example 1, “The first-last name style”* shows an example of a personal name to be formatted with the “first-last” style.

```
<para>The author is  
<personname role="first-last">  
  <givenname>Norman</givenname>  
  <surname>Walsh</surname>  
  <othername>David</othername>  
</personname>.</para>
```

### Example 1. The first-last name style

That would be formatted as:

The author is Norman David Walsh.

# t:person-name-last-first

Formats a personal name in the “last-first” style.

Template: {http://docbook.org/ns/docbook/templates}person-name-last-first

Defined in modules/info.xsl

Used in: modules/info.xsl

Used by: t:person-name

## Synopsis

```
<xsl:template name="t:person-name-last-first"/>
```

## Description

The “last-first” style formats a personal name as the family name (surname) followed by the first given name (if there is one). If both a family name and a given name are present, they are separated by a comma. See t:person-name .

*Example 1, “The last-first name style”* shows an example of a personal name to be formatted with the “last-first” style.

```
<para>The author is  
<personname role="last-first">  
    <givenname>Norman</givenname>  
    <surname>Walsh</surname>  
    <othername>David</othername>  
</personname>.</para>
```

### Example 1. The last-first name style

That would be formatted as:

The author is Walsh, Norman.

# t:person-name-list

Formats a sequence of personal names.

Template: {http://docbook.org/ns/docbook/templates}person-name-list

Defined in: modules/info.xsl

Used in: modules/bibliography.xsl

## Synopsis

```
<xsl:template name="t:person-name-list"/>
```

## Description

Formats a sequence of names:

- If the sequence consists of a single name, it is formatted.
- If the sequence consists of exactly two names, they are formatted with “and” between them where “and” is taken from the in-scope language localization file. It is the item with the key “author-sep2”.
- If the sequence consists of more than two names, all but the last are separated by “, “ (the localization item with the key “author-sep”). The last is separated by “, and” (the localization item with the key “author-seplast”).

# t:table-footnotes

Process table footnotes.

Template: {http://docbook.org/ns/docbook/templates}table-footnotes

Defined in: modules/footnotes.xsl

Used in: modules/tablecals.xsl, modules/tablehtml.xsl

## Synopsis

```
<xsl:template name="t:table-footnotes">
    <xsl:param name="footnotes" as="element(db:footnote)+"/>
</xsl:template>
```

## Description

The `t:table-footnotes` template is called to process footnotes at the bottom of a table.

# t:top-nav

Generate top-of-page navigation links.

Template: {http://docbook.org/ns/docbook/templates}top-nav

Defined in: modules/chunk-cleanup.xsl

Used in: modules/chunk-cleanup.xsl

## Synopsis

```
<xsl:template name="t:top-nav">
    <xsl:param name="chunk" as="xs:boolean"/>
    <xsl:param name="node" as="element()"/>
    <xsl:param name="prev" as="element()?" />
    <xsl:param name="next" as="element()?" />
    <xsl:param name="up" as="element()?" />
    <xsl:param name="top" as="element()?" />
</xsl:template>
```

## Description

This template generates the top-of-page `nav` element. When chunking is performed, this should provide links between pages.

## t:video-fallback

Generate fallback for video.

### Description

Video fallback is now supported with `$fallback-js`.

# t:xlink

Supports hypertext linking.

Template: {http://docbook.org/ns/docbook/templates}xlink

Defined in: modules/xlink.xsl

Used in: modules/links.xsl

## Synopsis

```
<xsl:template name="t:xlink">
    <xsl:param name="content"/>
</xsl:template>
```

## Description

In DocBook, most elements can be links. The `t:xlink` template is called for each element that might be participating in a link. This includes both simple links (any inline element with an `xlink:href` attribute) and extended links.



# V. Mode reference

# m:annotation-content

Mode for the content of annotations.

Mode: {http://docbook.org/ns/docbook/modes}annotation-content

Defined in: modules/annotations.xsl

## Description

Annotations, like footnotes, have markers and wrappers that are necessary to support their presentation. The `m:annotation-content` mode is used to process the actual body of the annotation.

# m:ansi

Mode for processing `funcsynopsis` elements in the “ANSI” style.

Mode: {http://docbook.org/ns/docbook/modes}ansi

Defined in: modules/programming.xsl (14)

## Description

The `funcsynopsis` element has several possible renderings. This mode is used when the “ANSI” style of rendering has been selected and a non-tabular layout is being used.

See also: `m:ansi-table` , `m:kr` , `m:kr-table` , `$funcsynopsis-default-style` , `$funcsynopsis-table-threshold` , and `$funcsynopsis-trailing-punctuation` .

# m:ansi-table

Mode for processing `funcsynopsis` elements in the “ANSI” style.

Mode: {http://docbook.org/ns/docbook/modes}ansi-table

Defined in: modules/programming.xsl (14)

## Description

The `funcsynopsis` element has several possible renderings. This mode is used when the “ANSI” style of rendering has been selected and a table will be used for layout.

See also: `m:ansi` , `m:kr` , `m:kr-table` , `$funcsynopsis-default-style` ,  
`$funcsynopsis-table-threshold` , and `$funcsynopsis-trailing-punctuation` .

# m:attributes

Mode for processing attributes.

Mode: {http://docbook.org/ns/docbook/modes}attributes

Defined in: modules/attributes.xsl (31)

## Description

Each DocBook element in the source document produces a primary HTML element in the result document. The attributes associated with that primary element are obtained by processing the DocBook element in the `m:attributes` mode.

# m:biblio690

Mode for processing ISO 690 `biblioentry` bibliographic entries.

Mode: {http://docbook.org/ns/docbook/modes}biblio690

Defined in: modules/biblio690.xsl (33)

Since: 2.0.1

## Description

DocBook bibliographic entries come in two forms: raw and cooked (<https://tdg.docbook.org/tdg/5.2/ch02.html#s.bibliography>). The `biblioentry` form is the “raw” form. When the `$bibliography-style` parameter is “`iso690`”, this mode is used to process `biblioentry` elements.

# m:biblioentry

Mode for processing `biblioentry` bibliographic entries.

Mode: `{http://docbook.org/ns/docbook/modes}biblioentry`

Defined in: `modules/bibliography.xsl` (11)

## Description

DocBook bibliographic entries come in two forms: raw and cooked (<https://tdg.docbook.org/tdg/5.2/ch02.html#s.bibliography>). The `biblioentry` form is the “raw” form. The `m:biblioentry` mode is used to process these elements.

# m:bibliomixed

Mode for processing `bibliomixed` bibliographic entries.

Mode: {http://docbook.org/ns/docbook/modes}bibliomixed

Defined in: modules/bibliography.xsl (7)

## Description

DocBook bibliographic entries come in two forms: raw and cooked (<https://tdg.docbook.org/tdg/5.2/ch02.html#s.bibliography>). The `bibliomixed` form is the “cooked” form. The `m:bibliomixed` mode is used to process these elements.

# m:callout-bug

Mode for producing callout marks.

Mode: {http://docbook.org/ns/docbook/modes}callout-bug

Defined in: modules/verbatim.xsl (3)

## Description

Callouts are processed in this mode to produce the callout marks (or “bugs”) used to indicate cross references between the callouts and their corresponding areas. This mode should only produce the mark, for example “①”. See [m:callout-link](#).

By default, callouts are numbered sequentially from 1 within each listing. However, a `db` processing instruction placed in the `areaspes` can specify a different starting number. If the `starting-callout-number` pseudo-attribute is an integer, that's used as the starting number. If it has the special value “`continues`”, then numbering continues sequentially after the last callout number used in the preceding verbatim listing.

# m:callout-link

Mode for producing callout links.

Mode: {http://docbook.org/ns/docbook/modes}callout-link

Defined in: modules/lists.xsl (2)

## Description

Callouts in a `calloutlist` can link back to the corresponding point in the listing or on the image. This mode is used to produce those links.

# m:chunk-cleanup

Post-process HTML chunks.

Mode: {http://docbook.org/ns/docbook/modes}chunk-cleanup

Defined in: modules/chunk-cleanup.xsl (15)

## Description

Every HTML chunk is processed with the `m:chunk-cleanup` mode. This mode makes sure that footnotes and annotations appear in the chunks where their references appear, adds navigation to the top and bottom of each chunk, etc. See *Section 2.6, ““Chunked” output”*.

# m:chunk-filename

Selects the chunk filename.

Mode: {http://docbook.org/ns/docbook/modes}chunk-filename

Defined in: modules/chunk.xsl (19)

## Description

Every element that is identified as a chunk, see *Section 2.6, ““Chunked” output”*, will be called in the `m:chunk-filename` mode to obtain the filename for that chunk.

# m:chunk-output

Performs the final-pass of processing on chunks.

Mode:	{http://docbook.org/ns/docbook/modes}chunk-output
Defined in:	modules/chunk-output.xsl (7)

## Description

Chunks (see [Section 2.6, ““Chunked” output”](#)) are constructed in several passes. The last pass is in the `m:chunk-output` mode. This mode makes last minute adjustments and produces the individual chunk documents.

# m:chunk-title

Generates the page title for the chunk.

Mode: {http://docbook.org/ns/docbook/modes}chunk-title

Defined in: modules/chunk-cleanup.xsl (4)

## Description

The page title, the `title` in the HTML `head`, is generated by processing the heading in this mode. This mode is responsible for removing markup from the title.

# m:chunk-write

Create result documents for chunks.

Mode: {http://docbook.org/ns/docbook/modes}chunk-write

Defined in: docbook.xsl

## Description

When all of the chunk processing is finished, each chunk is processed in `m:chunk-write` mode. By default, this mode creates XSL result documents for each chunk.

# m:copyright-years

Mode for processing copyright years.

Mode: {http://docbook.org/ns/docbook/modes}copyright-years

Defined in: modules/info.xsl

## Description

The `year` elements in a `copyright` are processed in this mode. This mode handles the separation (and possible collapsing) of years.

See also `$copyright-year-range-separator` and `$copyright-year-separator`, and `$copyright-collapse-years`.

# m:crossref

Produce a cross-reference to the element.

Mode: {http://docbook.org/ns/docbook/modes}crossref

Defined in: modules/xref.xsl

## Description

The `m:crossref` mode is used to generate a cross-reference.

## ~~m:crossref-inherit-separator~~

Produce the inherit separator.

Obsolete as of version 2.0.0.

## Description

Separators are now part of the generated text template, see *Chapter 4, Localization*.

# m:crossref-label

Produce the label in a cross-reference.

Mode: {http://docbook.org/ns/docbook/modes}crossref-label

Defined in: modules/xref.xsl (9)

## Description

The `m:crossref-label` mode is used to generate the label in a cross-reference.

# ~~m:crossref-label-separator~~

Produce the cross-reference label separator.

Obsolete as of version 2.0.0.

## Description

Separators are now part of the generated text template, see *Chapter 4, Localization*.

# m:crossref-number

Produce the number in a cross-reference.

Mode: {http://docbook.org/ns/docbook/modes}crossref-number

Defined in: modules/xref.xsl

## Description

The `m:crossref-number` mode is used to generate the number in a cross-reference.

# m:crossref-number-separator

Produce the cross-reference number separator.

Mode: {http://docbook.org/ns/docbook/modes}crossref-number-separator

Defined in modules/xref.xsl

## Description

The `m:crossref-number-separator` mode is used to generate the separator between a number and what follows it in a cross-reference.

## ~~m:crossref-prefix~~

Produce the cross-reference prefix.

Obsolete as of version 2.0.0.

## Description

Prefixes are now part of the generated text template, see *Chapter 4, Localization*.

# m:crossref-suffix

Produce the cross-reference suffix.

Mode: {http://docbook.org/ns/docbook/modes}crossref-suffix

Defined in: modules/xref.xsl

## Description

The `m:crossref-suffix` mode is used to generate any text that follows a cross reference.

# m:crossref-title

Produce the title in a cross-reference.

Mode: {http://docbook.org/ns/docbook/modes}crossref-title

Defined in: modules/xref.xsl (6)

## Description

The `m:crossref-title` mode formats the title in a cross-reference.

# m:details

Produce details about a media object.

Mode:	{http://docbook.org/ns/docbook/modes}details
Defined in:	modules/objects.xsl (2)

## Description

This mode is used to produce the details about a media object, such as a long description.

## ~~m:details-attribute~~

Produce the short summary for a media object.

Obsolete as of version 2.0.17.

## Description

This mode is used to produce a short detail summary about a media object.  
This is most often used as the alt text for the object.

# m:docbook

The primary mode for processing DocBook elements.

Mode: {http://docbook.org/ns/docbook/modes}docbook

Defined in main.xsl (2), modules/admonitions.xsl, modules/annotations.xsl (2),  
modules/attributes.xsl (4), modules/bibliography.xsl (5), modules/  
blocks.xsl (19), modules/components.xsl, modules/divisions.xsl, modules/  
footnotes.xsl (3), modules/glossary.xsl (10), modules/index.xsl (14),  
modules/info.xsl (15), modules/inlines.xsl (103), modules/l10n.xsl,  
modules/links.xsl (6), modules/lists.xsl (22), modules/msgset.xsl (7),  
modules/objects.xsl (12), modules/programming.xsl (24), modules/  
publishers.xsl (3), modules/refentry.xsl (11), modules/sections.xsl (2),  
modules/tablecals.xsl (8), modules/tablehtml.xsl (4), modules/  
templates.xsl, modules/titles.xsl, modules/toc.xsl (7), modules/  
unhandled.xsl (2), modules/verbatim.xsl (11), modules/xlink.xsl (4)

Used by: t:index-zone-reference, t:xlink, t:person-name

## Description

The `m:docbook` mode is the primary mode for processing DocBook elements. Critically, this means that if you import the DocBook stylesheets and wish to override some of the templates, you *must* make sure that those templates are defined in the `m:docbook` mode.

# m:footnote-number

Mode for generating footnote numbers.

Mode: {http://docbook.org/ns/docbook/modes}footnote-number

Defined in: modules/chunk-cleanup.xsl, modules/footnotes.xsl

## Description

A `footnote` processed in `m:footnote-number` mode renders the footnote number (or other symbol) associated with that footnote.

# m:footnotes

Mode for processing footnotes.

Mode: {http://docbook.org/ns/docbook/modes}footnotes

Defined in: modules/footnotes.xsl

## Description

Perhaps unsurprisingly, `footnote` elements are processed in `m:footnotes`. On the first pass, they're rendered inline with appropriate footnote markers. The “chunk cleanup” pass moves them to an appropriate location.

# m:generate-titlepage

Generate a title page.

Mode: {http://docbook.org/ns/docbook/modes}generate-titlepage

Defined in: modules/templates.xsl

## Description

All “block” elements that have a `title` (or another feature that constitutes a title) are processed in the `m:generate-titlepage` mode to generate their “title page”. Don’t be misled by the term “title page”. For many elements, the title page consists of nothing more than a single header element.

# m:gentext

Generate localization-specific text for an element.

Mode:	{http://docbook.org/ns/docbook/modes}gentext
Defined in:	modules/gentext.xsl (2)
Since:	2.0.0

## Description

This mode is used to generate text by applying the appropriate locale-specific template, see *Chapter 4, Localization*. When you apply templates in this mode, there are three parameters you can pass:

`group` (required)

This identifies the localization group.

`key`

This identifies the localization key. If not provided, the local name of the context item is used.

`content`

The content of the element. If not provided, the empty sequence is used.

For example, in the context of a book containing several chapters,

```
1 <xsl:apply-templates select="db:chapter[1]" mode="m:gentext">
  <xsl:with-param name="group" select=""title""/>
  <xsl:with-param name="content">
    <xsl:apply-templates select="db:chapter[1]/db:info/db:title/node()" />
  5 </xsl:with-param>
</xsl:apply-templates>
```

Would generate something like `Chapter 1 The Title`.

Lists are generated with the `m:gentext-list` mode.

# m:gentext-list

Generate localization-specific list for a sequence.

Mode:	{http://docbook.org/ns/docbook/modes}gentext-list
Defined in:	modules/l10n.xsl
Since:	2.0.0

## Description

This mode is used to generate a list with local-specific separators, see *Chapter 4, Localization*. When you apply templates in this mode, there are two parameters you can pass:

`list` (required)

A sequence of one or more items.

`name`

This identifies the localization key. If not provided, the local name of the context item is used.

For example, in the context of an `authorgroup`,

```
1 <xsl:apply-templates select=". " mode="m:gentext-list">
  <xsl:with-param name="list" as="item()"+>
    <xsl:apply-templates select="*"/>
  </xsl:with-param>
5 </xsl:apply-templates>
```

Would generate different markup depending on the number of authors. For example, in the English localization, it would generate:

Author Name

for a single author

Author Name and Second Author

for two authors

Author Name, Second Author, and Third Author

for three authors

Author Name, Second Author, Third Author, and Fourth Author

for four authors

And so on.

Generated text not associated with lists is generated with the `m:gentext` mode.

# m:headline

Produce a headline title.

Mode: {http://docbook.org/ns/docbook/modes}headline

Defined in: modules/titles.xsl

## Description

This mode produces the “headline” title for an element. This generally appears in the text where the element is formatted, but may also occur in the table of contents, the index, and other places.

# m:headline-label

Produce the label for a headline title.

Mode: {http://docbook.org/ns/docbook/modes}headline-label

Defined in: modules/titles.xsl (5)

## Description

The `m:headline-label` mode is used to generate the label in a headline.

# ~~m:headline-label-separator~~

Produce the headline label separator.

Obsolete as of version 2.0.0.

## Description

Separators are now part of the generated text template, see *Chapter 4, Localization*.

# m:headline-number

Produce the headline number.

Mode: {http://docbook.org/ns/docbook/modes}headline-number

Defined in: modules/titles.xsl (10)

## Description

The `m:headline-number` mode is used to generate the number in a headline.

# ~~m:headline-number-separator~~

Produce the headline number separator.

Obsolete as of version 2.0.0.

## Description

Separators are now part of the generated text template, see *Chapter 4, Localization*.

# m:headline-prefix

Generate the headline prefix.

Obsolete as of version 2.0.0.

## Description

Prefix text is now part of the generated text template, see *Chapter 4, Localization*.

## ~~m:headline-suffix~~

Produce the headline suffix.

Obsolete as of version 2.0.0.

## Description

Suffix text is now part of the generated text template, see *Chapter 4, Localization*.

# m:headline-title

Produce the headline title.

Mode: {http://docbook.org/ns/docbook/modes}headline-title

Defined in: modules/titles.xsl (6)

## Description

The `m:headline-title` mode is used to format the title element in a headline.

# m:highlight-options

Determine the syntax highlighting options for an element.

Mode: {http://docbook.org/ns/docbook/modes}highlight-options

Defined in: modules/verbatim.xsl

## Description

A verbatim element that will be syntax highlighted is processed in this mode to determine the options for the syntax highlighter. The template must return a map. The default template returns the language attribute if one is present.

# m:html-body-script

Add script elements to the end of a document.

Mode:	{http://docbook.org/ns/docbook/modes}html-body-script
Defined in:	modules/head.xsl
Deprecated since:	

## Deprecated

The technique described here is no longer necessary. Modern browsers accept a `defer` attribute on the `script` tag to explicitly specify when scripts are to execute.

## Description

One way to assure that JavaScript will only be invoked by the user agent after the DOM tree has been constructed is to put the script elements for that code at the end of the file.

The root element of each result document is formatted once in the `m:html-body-script` mode. Any elements constructed in this mode will be added to the end of the HTML `body`. The built-in implementation does nothing; this mode exists as an extensibility point. See also `m:html-head-script`.

# m:html-head

Construct the HTML `head` element.

Mode: `{http://docbook.org/ns/docbook/modes}html-head`

Defined in: `modules/head.xsl` (5)

## Description

The `m:html-head` mode is used to produce the `head` element. This should include all of the metadata, links, scripts, etc. necessary to render the document correctly.

It is not supposed to be necessary to override this template. You can customize what is produced with the `m:html-head-script`, `m:html-head-links`, and `m:html-head-last` modes.

The order of elements produced in the `head` is:

1. Elements generated by the stylesheets automatically (`title`, scripts, links, and other elements).
2. Elements generated in `m:html-head-script` mode.
3. Elements generated in `m:html-head-links` mode.
4. Elements in the HTML namespace that are in the source document's `info`.
5. Elements generated in `m:html-head-last` mode.

In addition, elements generated in `m:html-body-script` occur near the end of the HTML `body` element.

# m:html-head-last

Add elements to the end of the head of a document.

Mode: {http://docbook.org/ns/docbook/modes}html-head-last

Defined in: modules/head.xsl

## Description

Any elements constructed in `m:html-head-last` mode will be added to the very end of the HTML `head` element. The built-in implementation does nothing; this mode exists as an extensibility point.

# m:html-head-links

Add link elements to the head of a document.

Mode: {http://docbook.org/ns/docbook/modes}html-head-links

Defined in: modules/head.xsl

## Description

Any elements constructed in `m:html-head-links` mode will be added to the HTML `head` element. The built-in implementation does nothing; this mode exists as an extensibility point.

# m:html-head-script

Add link elements to the head of a document.

Mode: {http://docbook.org/ns/docbook/modes}html-head-script

Defined in: modules/head.xsl

## Description

Any elements constructed in `m:html-head-script` mode will be added to the HTML `head` element. The built-in implementation does nothing; this mode exists as an extensibility point.

# m:htmltable

Mode for processing HTML table elements.

Mode: {http://docbook.org/ns/docbook/modes}htmltable

Defined in: modules/tablehtml.xsl (2)

## Description

Where DocBook uses the HTML table model, it copies the definitions of the table elements (and the attributes on those elements) from HTML. The `m:htmltable` mode simply copies those elements and attributes to the result. When processing the contents of table cells, processing switches back to `m:docbook` mode.

# m:imagemap

Construct an HTML imagemap.

Mode: {http://docbook.org/ns/docbook/modes}imagemap

Defined in: modules/objects.xsl

## Description

The `imageobjectco` element is processed in this mode to construct an HTML image `map`.

# m:index-div

Mode for generating divisions in an index.

Mode: {http://docbook.org/ns/docbook/modes}index-div

Defined in: modules/index.xsl

## Description

Index processing, that is, the process of *generating* an index, involves collecting all of the `indexterm`s together, collating them, and rendering them.

The terms are grouped alphabetically and each group is processed in the `m:index-div` mode.

# m:index-primary

Mode for generating primary index entries.

Mode: {http://docbook.org/ns/docbook/modes}index-primary

Defined in: modules/index.xsl

## Description

When generating an index, each distinct `indexterm` is processed in `m:index-primary` mode to output its primary entry.

# m:index-secondary

Mode for generating secondary index entries.

Mode: {http://docbook.org/ns/docbook/modes}index-secondary

Defined in: modules/index.xsl

## Description

When generating an index, each distinct secondary `indexterm` is processed in `m:index-secondary` mode to output its secondary entry.

# m:index-see

Mode for generating index “see” entries.

Mode: {http://docbook.org/ns/docbook/modes}index-see

Defined in: modules/index.xsl

## Description

When generating an index, each `indexterm` that defines a “see” index cross-reference is processed in `m:index-see` mode to output the cross-reference.

# m:index-seealso

Mode for generating index “see also” entries.

Mode: {http://docbook.org/ns/docbook/modes}index-seealso

Defined in: modules/index.xsl

## Description

When generating an index, each `indexterm` that defines a “see also” index cross-reference is processed in `m:index-seealso` mode to output the cross-reference.

# m:index-tertiary

Mode for generating index tertiary entries.

Mode: {http://docbook.org/ns/docbook/modes}index-tertiary

Defined in: modules/index.xsl

## Description

When generating an index, each distinct tertiary `indexterm` is processed in `m:index-tertiary` mode to output its tertiary entry.

# m:kr

Mode for processing `funcsynopsis` elements in the “K&R” style.

Mode: {http://docbook.org/ns/docbook/modes}kr

Defined in: modules/programming.xsl (14)

## Description

The `funcsynopsis` element has several possible renderings. This mode is used when the “K&R” style of rendering has been selected and a non-tabular layout is being used.

See also: `m:kr-table` , `m:ansi` , `m:ansi-table` , `$funcsynopsis-default-style` , `$funcsynopsis-table-threshold` , and `$funcsynopsis-trailing-punctuation` .

# m:kr-args

Mode for processing function arguments.

Mode:	{http://docbook.org/ns/docbook/modes}kr-args
Defined in:	modules/programming.xsl (7)

## Description

This mode is used to process the parameters of a `funcsynopsis` in th K&R style.  
See `$funcsynopsis-default-style`.

# m:kr-table

Mode for processing `funcsynopsis` elements in the “K&R” style.

Mode: {http://docbook.org/ns/docbook/modes}kr-table

Defined in: modules/programming.xsl (14)

## Description

The `funcsynopsis` element has several possible renderings. This mode is used when the “K&R” style of rendering has been selected and a table will be used for layout.

See also: `m:kr` , `m:ansi` , `m:ansi-table` , `$funcsynopsis-default-style` , `$funcsynopsis-table-threshold` , and `$funcsynopsis-trailing-punctuation` .

# m:kr-table-args

Mode for processing function arguments in a table.

Mode: {http://docbook.org/ns/docbook/modes}kr-table-args

Defined in: modules/programming.xsl (7)

## Description

When the tabular presentation of a “K&R” `funcsynopsis` is being generated, this mode is used to process the function arguments.

# m:link

Process inline links.

Mode:	{http://docbook.org/ns/docbook/modes}link
Defined in:	modules/links.xsl (2)

## Description

DocBook allows `xlink:href` to appear on most elements. All inline elements are processed in `m:link` mode to add a link around them if they specify an `xlink:href`.

# m:list-of-equations

Mode for generating a list of equations.

Mode: {http://docbook.org/ns/docbook/modes}list-of-equations

Defined in: modules/toc.xsl (2)

## Description

If `$lists-of-equations` is true, then a list of equations will be generated. This mode is used to process the `equation` elements (if any) to produce that list.

# m:list-of-examples

Mode for generating a list of examples.

Mode: {http://docbook.org/ns/docbook/modes}list-of-examples

Defined in: modules/toc.xsl (2)

## Description

If `$lists-of-examples` is true, then a list of examples will be generated. This mode is used to process the `example` elements (if any) to produce that list.

# m:list-of-figures

Mode for generating a list of figures.

Mode: {http://docbook.org/ns/docbook/modes}list-of-figures

Defined in: modules/toc.xsl (2)

## Description

If `$lists-of-figures` is true, then a list of figures will be generated. This mode is used to process the `figure` elements (if any) to produce that list.

# m:list-of-procedures

Mode for generating a list of procedures.

Mode: {http://docbook.org/ns/docbook/modes}list-of-procedures

Defined in: modules/toc.xsl (2)

## Description

If `$lists-of-procedures` is true, then a list of procedures will be generated. This mode is used to process the `procedure` elements (if any) to produce that list.

# m:list-of-tables

Mode for generating a list of tables.

Mode: {http://docbook.org/ns/docbook/modes}list-of-tables

Defined in: modules/toc.xsl (2)

## Description

If `$lists-of-tables` is true, then a list of tables will be generated. This mode is used to process the `table` elements (if any) to produce that list.

# m:list-of-titles

Mode for generating a list-of-titles.

Mode: {http://docbook.org/ns/docbook/modes}list-of-titles

Defined in: modules/toc.xsl

Since: 2.0.0

## Description

When a list of titles (figures, tables, equations, etc.) is created, each element is processed in this mode to produce the list item that will appear in the list.

# m:mediaobject-end

Process the end of media objects.

Mode: {http://docbook.org/ns/docbook/modes}mediaobject-end

Defined in: modules/objects.xsl (3)

## Description

Media objects (both the `mediaobject` and `inlinemediaobject` containers as well as the `imageobject`s etc. inside them) are processed in this mode after the primary content has been rendered.

For `mediaobject`, this mode is responsible for outputting the `caption`, but may also be used to output accessibility details, legal notices, etc. See also *m:mediaobject-start*.

# m:mediaobject-info

Compute properties for a media object.

Mode: {http://docbook.org/ns/docbook/modes}mediaobject-info

Defined in: modules/objects.xsl (4)

## Description

The media object elements are processed in this mode to extract their properties (input and output URIs, content types, etc.). See *Section 5.4, “Processing mediaobjects”*.

# m:mediaobject-output-adjust

Adjust the URI references to media objects.

Mode: {http://docbook.org/ns/docbook/modes}mediaobject-output-adjust

Defined in modules/chunk-cleanup.xsl

## Description

Once the stylesheets have used the `$mediaobject-input-base-uri`, `$mediaobject-output-base-uri`, and `$mediaobject-output-paths` to compute the URI of a media reference, the reference is processed in `m:mediaobject-output-adjust` mode. This is the stylesheet's opportunity to make any final adjustments.

The context item for the template will be the attribute that contains the author's original value. The adjusted value is passed in as the `$adjusted-uri` parameter. The value returned by the template is used in the HTML.

Here is an example that groups images, audio, and video files in their own directories:

```

1 <xsl:template match="@*" mode="m:mediaobject-output-adjust">
  <xsl:param name="adjusted-uri" as="xs:string"/>

  <xsl:choose>
5   <xsl:when test="exists(f:uri-scheme(.))">
     <!-- Don't mess with absolute URLs... -->
     <xsl:sequence select="$adjusted-uri"/>
   </xsl:when>
   <xsl:otherwise>
10  <xsl:variable name="type" as="xs:string">
    <xsl:choose>
      <xsl:when test=".//self::h:img">image</xsl:when>
      <xsl:when test="ancestor::h:video">video</xsl:when>
      <xsl:when test="ancestor::h:audio">audio</xsl:when>
15  <xsl:otherwise>
      <xsl:sequence select=""media-cleanup-err""/>
    </xsl:otherwise>
    </xsl:choose>
    </xsl:variable>
20
20  <xsl:variable name="parts" select=" tokenize($adjusted-uri, '/')"/>
  <xsl:sequence select="string-join($parts[position() lt last()], '/')
    || (if (count($parts) gt 1) then '/' else ")
    || $type || '/'
25  || $parts[position() eq last()]">
  </xsl:otherwise>
</xsl:choose>
</xsl:template>

```

In other words, if the adjusted URI for an image is `path/to/somewhere.png`, this template will return `path/to/image/somewhere.png` and make similar adjustments to the audio and video paths.

# m:mediaobject-start

Process the start of media objects.

Mode: {http://docbook.org/ns/docbook/modes}mediaobject-start

Defined in: modules/objects.xsl (4)

## Description

Media objects (both the `mediaobject` and `inlinemediaobject` containers as well as the `imageobject`s etc. inside them) are processed in this mode immediately after the start tag. Templates in this mode can output attributes, such as `summary` or `alt`, as well as additional accessibility elements; it may also be used to output legal notices, etc. that should precede the primary content.

If the `$mediaobject-accessibility` parameter includes `a11y-metadata` then any short accessibility summary will be output in a `meta` element with the property value `a11y:accessibilitySummary`. In addition, any `meta` elements in the `info` will be processed in this mode as well (so they can also be output using accessibility meta properties. See also *m:mediaobject-end*.

# m:mediaobject-uris

Compute URIs for media objects.

Mode: {http://docbook.org/ns/docbook/modes}mediaobject-uris

Defined in: modules/objects.xsl

## Description

The media object elements are processed in this mode to extract their input and output URIs.

# m:persistent-toc

Mode for generating the persistent table of contents.

Mode: {http://docbook.org/ns/docbook/modes}persistent-toc

Defined in: modules/toc.xsl (3)

Since: 2.0.0

## Description

If `$persistent-toc` is true, the document will be processed in this mode to produce the contents of the persistent ToC. By default, this produces the same ToC as the `m:toc` mode.

# m:production-number

Mode for production numbers.

Mode: {http://docbook.org/ns/docbook/modes}production-number

Defined in: modules/programming.xsl

## Description

When a `productionset` is being formatted, the `m:production-number` mode is used to produce the production numbers.

# m:pygments-options

Calculate the Pygments options for an element.

Mode: {http://docbook.org/ns/docbook/modes}pygments-options

Defined in: modules/verbatim.xsl

## Description

When the Pygments syntax highlighter is being used, every element that will be highlighted is processed in this mode to generate options for the Pygments process.

# m:revhistory-list

Mode to generate a “list” presentation of `revhistory`.

Mode: `{http://docbook.org/ns/docbook/modes}revhistory-list`

Defined in: `modules(blocks.xsl)`

## Description

The `revhistory` element can be rendered as a list or a table (see `$revhistory-style`). The `m:revhistory-list` mode is used to render it as a list.

# m:revhistory-table

Mode to generate a “table” presentation of `revhistory`.

Mode: `{http://docbook.org/ns/docbook/modes}revhistory-table`

Defined in: `modules/blocks.xsl`

## Description

The `revhistory` element can be rendered as a list or a table (see `$revhistory-style`). The `m:revhistory-table` mode is used to render it as a list.

# m:seglist-table

Process a `segmentedlist` as a table.

Mode: {http://docbook.org/ns/docbook/modes}seglist-table

Defined in: modules/lists.xsl (4)

## Description

Segmented lists can be presented in either a list format or a table format. If the tabular format is selected, the `segmentedlist` is processed in this mode to generate the table.

# m:segtitle-in-seg

Mode for processing segment titles in a segment.

Mode: {http://docbook.org/ns/docbook/modes}segtitle-in-seg

Defined in: modules/lists.xsl

## Description

In a `segmentedlist`, each `segtitle` is processed once for each segment. This processing occurs in the `m:segtitle-in-seg` mode.

# m:synopfragment-bug

Mode for generating the mark for a `synopfragmentref`.

Mode: `{http://docbook.org/ns/docbook/modes}synopfragment-bug`

Defined in: `modules/programming.xsl`

## Description

The `synopfragment` element allows a synopsis (for example a `cmdsyntax`) to move common fragments out-of-line. The location where these fragments occur in the primary synopsis is indicated with a `synopfragmentref`. The stylesheets will generate a mark, analogous to the marks used for footnotes or callouts, to connect the reference to the fragment. The fragment will be processed in `m:synopfragment-bug` to produce that mark.

# m:synopsis

Mode for processing programming language synopses.

Mode:	{http://docbook.org/ns/docbook/modes}synopsis
Defined in:	modules/programming.xsl (10)

## Description

The elements of class, method, and field synopses are processed in this mode.

# m:title

Format a title.

Mode:	{http://docbook.org/ns/docbook/modes}title
Defined in:	modules/titles.xsl

## Description

This mode is used to format titles when they appear in headlines or cross references.

# m:titlepage

Format elements for the title page.

Mode:	{http://docbook.org/ns/docbook/modes}titlepage
Defined in:	modules/titlepage.xsl (9)

## Description

Elements that appear on the title page are processed in this mode to produce the text that will appear on the title page.

# m:to-uppercase

Produce content in upper-case.

Mode: {http://docbook.org/ns/docbook/modes}to-uppercase

Defined in: modules/info.xsl (4)

## Description

When formatting personal names in a style that requires converting a name to upper case (see *t:person-name*), this mode is used to generate the upper-case name.

# m:toc

Mode for generating lists of titles (e.g, the table of contents)

Mode: {http://docbook.org/ns/docbook/modes}toc

Defined in: modules/blocks.xsl (4), modules/toc.xsl (2)

## Description

Elements are processed in `m:toc` when automatically constructing a Table of Contents (or, more generally, any list of titles).

Where a generated list of titles appears is controlled by several parameters. For manual control, place an empty `toc` element where you would like it to appear. If the `toc` element is not valid where you would like the list to appear, use the `db-toc` processing instruction instead. These must appear as the direct children of the element for which the list is being constructed.

See also `$generate-toc`, `$generate-nested-toc`, `$annotate-toc`, `$lists-of-figures`, `$lists-of-tables`, `$lists-of-examples`, `$lists-of-equations`, `$lists-of-procedures`, and `$section-toc-depth`.

# m:toc-entry

Format content for a list-of-titles.

Mode: {http://docbook.org/ns/docbook/modes}toc-entry

Defined in: modules/toc.xsl (6)

## Description

When an element appears in the Table of Contents (or other list of titles), it is formatted in this mode to produce the entry in the list.

# m:toc-nested

What elements are “nested” in a Table of Contents.

Mode:	{http://docbook.org/ns/docbook/modes}toc-nested
Defined in:	modules/toc.xsl (4)
Since:	2.0.0

## Description

The elements processed in the `m:toc` mode determine where a Table of Contents will appear. When a Table of Contents does appear, the children of the starting element are processed in the `m:toc-nested` mode to determine which of them appear. This mode, for example, handles the maximum nesting depth of sections in the ToC.

# VI. Processing instruction reference

# DocBook-xslTNG-version

Resolves to the DocBook stylesheet version.

Processing instruction:DocBook-xslTNG-version

## Description

The `DocBook-xslTNG-version` processing instruction will be replaced by the version number of the stylesheets.

For example:

```
<para>This document was formatted with the <citetitle>DocBook  
xslTNG Stylesheets</citetitle>  
version <?DocBook-xslTNG-version?>.</para>
```

Will render like this:

This document was formatted with the *DocBook xslTNG Stylesheets* version 2.1.0.

# current-dateTime

Resolves to a formatted date/time string.

Processing instruction:	current-dateTime
-------------------------	------------------

## Description

The `current-dateTime` processing instruction will be replaced by a formatted date and time. The date and time is controlled by three pseudo-attributes:

`dateTime`

The *ISO 8601* date or date/time to format. If not specified, defaults to the current date/time. If a date (and not a date/time) is specified, a default time of noon is assumed.

`offset`

An *ISO 8601* day/time duration or year/month duration. This value will be added to the `dateTime` that is to be formatted.

`format`

The format string to use. If not specified, the `$date-dateTime-format` is used.

For example:

```
<para>Published on  
<?current-dateTime format="[F]"?>.</para>
```

Will render like this:

Published on Friday.

# db

Provides additional control over rendering.

Processing instruction:

db

## Description

The `db` processing instruction provides additional control over the rendering of a number of elements. It doesn't produce any output directly.

See also:

- `$align-char-default`, `$align-char-pad`, `$align-char-width`
- `$bibliography-style`
- `$mediaobject-video-element`
- `$oxy-markup`
- `$revhistory-style`
- `$segmentedlist-style`
- `$xlink-style`
- `f:chunk-filename`
- `f:date-format`
- `f:pi`
- `f:pi-attributes`
- `f:verbatim-highlight`
- `f:verbatim-numbered`
- `f:verbatim-style`
- `f:verbatim-trim-trailing`
- `t:generate-index`
- `m:callout-bug`

# system-property

Resolves to an XSLT system property.

Processing instruction:	system-property
-------------------------	-----------------

## Description

The `system-property` processing instruction will be replaced by the corresponding system property.

For example:

```
<para>This document was formatted with  
<?system-property xsl:product-name?>  
version <?system-property xsl:product-version?> on  
<?current-dateTime format="[D01] [MNn,*-3] [Y0001]"?>.  
</para>
```

Will render like this:

This document was formatted with SAXON version HE 11.5 on 07 Apr 2023.



## Appendix B. GNU Free Documentation License

GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

### 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

### 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

*DocBook xslTNG*

Copyright © 2020–2023 Norman Walsh.  
All Rights Reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation. A copy of the license is included in *Appendix B, GNU Free Documentation License*.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and the authors were aware of a trademark claim, the designations have been printed in caps or initial caps.

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